PRAGMA

A publication of Semaphore Corporation

Issue Number 2 November, 1982

IN THIS ISSUE

DEPARTMENTS

PRAGMA

Issue Number 2 November, 1982

Pragma is published at least four times a year by

Semaphore Corporation
207 Granada Drive
Aptos, California 95003

SUBSCRIPTIONS

Subscriptions are $25 per issue, $100 per year (four issues) in the USA, $38 per issue to other countries by airmail. All payments must be in US dollars drawn on a US bank.

Paid subscribers who refer new subscribers will receive a free subscription extension of one issue for each referral. Such referrals must include an appropriate subscriber number as explained on subscription order forms.

Address all subscription correspondence to the **Pragma** Circulation Manager, **Semaphore Corporation.** When writing, enclose your issue's mailing address label or the numbers from the upper right corner of your label.

Address changes should be sent at least four weeks in advance. Include old and new addresses along with your issue's mailing address label.

SUBMITTALS

All correspondence and material received will be considered for publication. Except for correspondence used in the Letters Department, authors are paid up to $200 per full published page for submitted material used in **Pragma.** Actual payment amounts are decided by the Editors and vary with the amount of required editing and rework and with the length of each submittal. Authors are also granted free subscription extensions of one issue for submittals of at least one full published page. Address all submittals and correspondence to the **Pragma** Editors, **Semaphore Corporation.** All letters to the Editors are welcome and as many as possible will be published in the Letters Department.

All submittals must be typed on white paper and double spaced. The first page of any submittal and any accompanying material must include the author's name, address, telephone number and the date. All pages must be numbered.

Hand-typed program listings and other simulated printouts will not be accepted. Authors should submit actual computer-generated output. Print all listings with a fresh black ribbon on continuous white paper. Do not print on perforations. Do not include page numbers or headings on listings in order to simplify reduction and layout. Accompanying documentation should refer to listings by content, line number or symbolic labels, not by page number.

Drawings, schematics and other illustrations must be in black ink on white paper and drawn in a large scale to allow significant reduction. Photographs must be black and white glossies.

Manuscripts are submitted at the author's risk. Unused manuscripts will be returned if a stamped, self-addressed envelope is included. Requests to review galleys must accompany the manuscript when it is first submitted. The Editors reserve the right to edit all submittals.

ADVERTISING

Send all advertising correspondence, requests for advertising rates, and advertising copy to the **Pragma** Advertising Manager, **Semaphore Corporation.** Advertisers sending press releases are requested to telephone the Advertising Manager for an interview at 408-688-9200 within two weeks after submitting the material.

READER SERVICES

Is there a service **Semaphore Corporation** can provide for you? Address all inquiries to **Pragma** Reader Service, **Semaphore Corporation,** or telephone 408-688-9200.

We Have Liftoff

Launching **Pragma** has been a very rewarding experience. We've particularly enjoyed the many letters and especially the telephone calls from Pick users everywhere. One reader even dropped by while going jogging! From a number of conversations with subscribers, we've been able to glean a few impressions of our premiere issue:

- **Pragma**'s technical content and accuracy get high marks among readers.
- The Benchmarks and Wish List departments are very popular, and there is a lot of interest in seeing benchmark statistics comparing various manufacturers.
- An answer was missing in the Queries department. (Yes, that was deliberate. We don't always have an answer for every inquiry. Readers should feel free to submit any answers they have for questions left unresolved in the Queries department.)
- The Games department's program is usually the first **Pragma** code a reader will transport onto some computer.

Readers will notice a number of new developments in the look and content of this month's **Pragma.** First of all, the cover has been trimmed in red. With each issue, a different color will be used to highlight the cover and accent various ads throughout the journal. Having the cover color change with each issue will help readers identify and differentiate **Pragma** issues at a glance, especially when issues are retained and collected for reference purposes.

Also sprinkled throughout this issue are the names of just a few of the many new **Pragma** subscribers who have joined us since August. Space limitations prevent us from including every name in one issue, so future issues will also be featuring the names of our readers, until we have listed every subscriber who checked off the "YES, publish my name" box on their subscription form. (To our surprise, a large number of the YES boxes were not marked.)

This month, readers will also discover no less than three ads for financial modeling and spread sheet systems: Auroplan from Aurotech, Compusheet from Interactive Systems, and Cargo from Semaphore. The famous Visicalc spread sheet program originally developed for Apple microcomputer systems has become perhaps the most successful software package of all time, with spinoffs, variations and imitations flooding every part of the software market. This year, the flood raced into the Pick world, as evidenced by this issue's ads.

A number of articles have been submitted to us over the past months, and two are being prepared for our next issue. Many of the submittals we received included listings that had to be regenerated because the printed characters were too light. If you are a prospective author about to send in a listing, be sure it is printed with a fresh, black ribbon.

We're very happy with the enthusiastic reception that **Pragma** has received, and we appreciate hearing from our readers at all times. Don't hesitate to call or drop us a line.

—The Editors [P]

Pragma Is Looking for Submittals

Do you have an application program you would like to share? Have you discovered a bug in your system software? Do you have a question about your system for which you haven't been able to find an answer? Have you purchased a software product and written an evaluation of it? Have you performed benchmark tests and recorded the results? Perhaps you've written a small utility that has proven useful? Or maybe you've even programmed a game?

If you answered YES to any of the above, then you're on your way to being the author of an article in **Pragma.** Authors are awarded payments of up to $200 per full published page for submitted material used in **Pragma.** See the details under SUBMITTALS on the opposite page.

Send your material to the **Pragma** Editors, 207 Granada Drive, Aptos, California, 95003. [P]

A Modulo Setting Program

Software and techniques are described for the reallocation of file modulos. A program for placing the modulo parameter in file definition items is presented.

The primary method most installations use to optimize file allocation and keep system throughput at a maximum is to keep all file modulos properly adjusted. Files with good modulo settings will minimize frame faulting and avoid time-consuming reads and writes to the disk. Files with poor modulo settings will cause excessive frame faulting and thrashing and can quickly force a system to run intolerably slow. For these reasons, all files on disk should be periodically and frequently inspected for proper modulo settings. Any file with a poorly defined modulo should be reallocated with a better modulo to help keep system performance at an optimum.

There are two basic techniques for reallocating the modulo of an existing file: file copying and file restoring.

Reallocation of file modulos by file copying is accomplished with the use of a number of system commands. For example, if our existing file is called OLD, we can use the CREATE-FILE verb to create a new, temporarily named file with a newly selected modulo. Let's assume the temporary file is called NEW. COPY commands can then be used to copy all OLD dictionary and data items into the NEW file. The DELETE-FILE verb is then used to completely delete the OLD file, and one final COPY command can delete the file definition item for NEW and rename (copy) it back to the OLD file definition item.

This "brute force" technique works, and is frequently useful for quick adjustment of one or two files, especially if there is any kind of need to reallocate files in a hurry. But the method does have a number of disadvantages which make the procedure awkward or even dangerous. For instance, measures must be taken to insure that no other users are accessing the file while it is being reallocated, since pointers and data are being copied and deleted. Another problem is that the technique involves a number of commands even for adjusting one file, and is typically slow and prone to operator error. Also, a good number of disk frames may be consumed while temporarily allowing two copies of one large file to exist (even if items are immediately deleted while being copied from OLD to NEW). An installation could develop some sort of proc that automatically does the complete copy and delete procedure for every file that needs a new modulo, perhaps with the proc set to SLEEP until some evening period when no user activity will occur, but most sites simply take advantage of the file reallocation powers built into the system's file restore procedure as an alternate and easier way of adjusting modulos.

As explained in standard system documentation, the file restore procedure can automatically reallocate any file that has a new modulo specified in attribute thirteen of the file's definition item. The disadvantage to this approach is that a file's modulo is adjusted only after doing a file save and restore. But since many installations typically schedule a file save every day and a file restore once a week or so, files never go for very long before being reallocated. Periodic file restores not only facilitate the adjustment of file modulos, but also verify that the tapes used are readable, while compacting the allocation of frames on disk.

If file modulos are being reallocated during a file restore, then some mechanism must be available to place the new modulo in attribute thirteen of the definition item of any file being adjusted, before the file save and restore take place. If only a small number of files are being reallocated, then the system's text editor alone is a sufficient tool for editing attribute thirteen in any definition item. However, users often discover that a significant number of files need adjustment by the time the next file restore rolls around, and so it becomes desirable to have a program for automatically setting file modulo parameters in attribute thirteen of file definition items.

SET.MODULOS is the name of just such a program (written in DATA/BASIC for a Microdata), and is listed on page 7. SET.MODULOS is designed to read items from the system's STAT-FILE in order to determine the location and status of an existing file. SET.MODULOS then accesses the file's definition item and places a new, more appropriate modulo in attribute thirteen, to allow proper reallocation of the file during the next file restore.

Most of the code in SET.MODULOS is concerned with the accessing and updating of attribute thirteen in a D-pointer. The actual value (new modulo) to be placed in attribute thirteen can be determined with the use of any number of available algorithms concerned with computing proper modulos and separations. The algorithm that SET.MODULOS currently uses happens to be the same as that presented in the article *Computing Modulos with ENGLISH* (Pragma #1, page 29), but SET.MODULOS can easily be changed to use any other modulo calculation method.

Selecting STAT-FILE items is a convenient method for accessing the statistics on any file that needs reallocating, but probably not every file will need adjustment, so only some STAT-FILE items will be of interest. Selecting only those STAT-FILE items corresponding to files needing new modulos can be accomplished two ways: the program for updating attribute thirteen can use a DATA/BASIC SELECT statement and READ every STAT-FILE item, then ignore any with a correct modulo (where "correct" is determined by whatever algorithm for computing modulos happens to be im-

SET.MODULOS PROGRAM LISTING

```
001 OPEN "STAT-FILE" TO STAT.FILE ELSE STOP "ERR1", "STAT-FILE"
002 OPEN "MD" TO MD.FILE ELSE STOP "ERR1", "MD"
003 *
004 100 *
005 READNEXT STAT.ID ELSE
006   DELETE MD.FILE, "QFILE"
007   STOP
008   END
009 READ STATS FROM STAT.FILE, STAT.ID ELSE STOP "ERR2"
010 NAMES = STATS<1>
011 ACCOUNT = OCONV(NAMES, "G*1")
012 FILE = OCONV(NAMES, "G1*1")
013 LEVEL = STATS<2>
014 BEGIN CASE
015 CASE LEVEL = 0
016   DITEM.ID = ACCOUNT
017 CASE LEVEL = 1
018   DITEM.ID = ACCOUNT ; ACCOUNT = "SYSTEM" ; FILE = ""
019 CASE LEVEL = 2
020   DITEM.ID = FILE ; FILE = ""
021 CASE LEVEL = 3
022   DITEM.ID = "DL/ID"
023 CASE 1
024   GO TO 100
025 END CASE
026 QITEM = "Q"
027 QITEM<2> = ACCOUNT
028 QITEM<3> = FILE
029 QITEM<9> = "L"
030 QITEM<10> = "10"
031 WRITE QITEM ON MD.FILE, "QFILE"
032 OPEN "DICT","QFILE" TO Q.FILE ELSE STOP "ERR1", "QFILE"
033 READU DITEM FROM Q.FILE, DITEM.ID ELSE RELEASE ; GO TO 100
034 ITEMS = STATS<6>
035 SIZE = STATS<7>
036 NEW.MOD = INT(SIZE/500)
037 IF MOD(SIZE,500) # 0 THEN NEW.MOD = NEW.MOD+1
038 IF ITEMS < NEW.MOD THEN NEW.MOD = ITEMS
039 LOOP UNTIL (MOD(NEW.MOD,2) # 0) AND (MOD(NEW.MOD,5) # 0) DO
040   NEW.MOD = NEW.MOD+1
041 REPEAT
042 IF DITEM<13> = "" THEN DITEM<13> = "(":NEW.MOD:",1)" ELSE
043   DITEM<13> = DITEM<13>[1,INDEX(DITEM<13>,"(",1)]:NEW.MOD:",1)"
044   END
045 PRINT ; PRINT
046 PRINT ACCOUNT:"*":FILE:"*":DITEM.ID
047 PRINT
048 FOR I = 1 TO 13
049   PRINT STR("0",3-LEN(I)):I:" ":DITEM<I>
050 NEXT I
051 PRINT
052 LOOP
053   PRINT "File (FI) or exit (EX)":
054   INPUT ANSWER
055 UNTIL (ANSWER = "FI") OR (ANSWER = "EX") DO REPEAT
056 IF ANSWER = "FI" THEN WRITE DITEM ON Q.FILE, DITEM.ID ELSE RELEASE
057 *
058 GO TO 100
059 *
060 END
```

bedded in the program), or an ENGLISH SELECT can be used to find only the STAT-FILE items of interest, which are then passed to the update program. There are pros and cons for each method.

If a DATA/BASIC SELECT statement is used from within the update program, then the program is completely self-contained and just about any modulo algorithm can easily be used. One disadvantage of this approach is that if the program interacts with the user (typically to allow approval of the new modulo selected by the program), then there is often an annoying delay of unpredictable length while the program searches for the next STAT-FILE item needing attention. Also, STAT-FILE items can't be SELECTed in any particular order without resorting to external SELECTs.

If an external SELECT is used to feed the program, then STAT-FILE items can be chosen in any order, such as by account. This may help guide an operator interacting with the program, or help in the design of meaningful output messages that announce what type of progress the update program is making through all accounts on the system. If the external SELECT is intelligent enough to skip any STAT-FILE items not needing new modulos, annoying delays between file items are avoided once the program starts executing, although the SELECT itself may be quite time consuming. Also, it may be difficult to create an ENGLISH SELECT that is intelligent enough to duplicate the modulo computation algorithm being used. Even if an external SELECT can be used, two separate commands or pieces of software must work in conjunction, which is often more difficult to maintain than one self-contained program.

As presented here, SET.MODULOS is designed to work with an external SELECT. For example, this proc

```
001 PQN
002 OYou are about to
003 Oedit file modulos!
004 OIf you change items
005 Oin the SYSTEM file,
006 Othen no one else
007 Oshould be logged on!
008 OContinue+
009 IP?
010 IF A # YES RTN
011 HSSELECT STAT-FILE
012 H WITH FRAMES.
013 H AND WITH DELTA # "0"
014 H BY ACCOUNTNAME
015 H BY FILENAME.
016 STON
017 HRUN BP SET.MODULOS
018 P
019 RTN
```

is ideal for invoking the SET.MODULOS program. The dictionary words FRAMES., ACCOUNTNAME and FILENAME. are all standard STAT-FILE definitions, while DELTA should be defined as shown in Pragma #1. The proc selectively avoids any STAT-FILE items with no frame count and any which do not need a new modulo as determined by the DELTA word, while sorting the resulting item list by account and then by file, for input to the SET.MODULOS program. If desired, the AND WITH DELTA phrase can be left out of the proc and the SET.MODULOS program can be modified to make the determination of which STAT-FILE items indicate a new modulo is necessary. Or, the whole proc can be abandoned and a SELECT statement used from within the SET.MODULOS program.

The program begins execution in line 1 by OPENing the STAT-FILE (which means SET.MODULOS is usually executed from the SYSPROG account) and then the Master Dictionary, since a Q-pointer item named QFILE will later have to be placed there (which means SET.MODULOS must be executed with at least SYS1 privileges and the MD item named QFILE must be available for use by the program). If a SELECT STAT.FILE statement is going to be included in SET.MODULOS, it should be positioned after line 1 but before line 4.

Line 5 gets the next item identifier, otherwise it STOPs the program at line 7 if there are no more STAT-FILE items to process. Before STOPing, line 6 DELETEs any Q-pointer item left in the MD file by the program. To insure that using the name QFILE is allowed, it may be desirable to modify SET.MODULOS to READ the item named QFILE from the MD file somewhere before line 4, and then STOP if the READ succeeds, to avoid overwriting any previously existing MD item named QFILE.

Line 9 READs the actual STAT-FILE item, and lines 10 through 12 extract the account and file names that the statistics are for. Since the pointer for the file being adjusted can be in the SYSTEM file, or in a Master Dictionary, or in the DICT level of a file, the CASE statement in lines 13 through 25 uses attribute 2 of the STAT-FILE item to determine at exactly what level in the file system hierarchy the pointer will be found. The ACCOUNT, FILE and DITEM.ID variables are adjusted accordingly. DITEM.ID will be the name of the item that contains the attribute (thirteen) where the new modulo will be placed.

Lines 26 through 30 create a pointer to the file containing the definition item, and line 31 WRITEs the pointer to the MD file using the item name QFILE. Now that a pointer to the necessary file exists, line 32 can OPEN the file for the READU by line 33, which retrieves the D-pointer file definition item named by the DITEM.ID variable set by the previously mentioned CASE statement. Since the READU and subsequent WRITEs often cross over into other accounts, the account running SET.MODULOS (typically SYSPROG) should have all necessary update and retrieval locks defined in the SYSTEM file to allow access to other accounts, or the program will abort because of invalid access protection codes.

By line 34, DITEM contains the D-pointer that needs a new attribute thirteen. More statistics are taken from the STAT-FILE item in lines 34 and 35, and lines 36 to 41 (the modulo computation algorithm) are then executed to compute NEW.MOD, the new modulo to be planted in attribute thirteen of DITEM. If a SELECT is being done from within SET.MODULOS, then it is probably better to move the modulo computation up to somewhere just after line 9, and a GOTO 100 can then be executed to skip any item not needing adjustment if NEW.MOD is found to still equal the old modulo, before bothering with the QFILE item.

Lines 42 to 44 place the new modulo in attribute thirteen, being careful not to disturb any special flag characters such as those found in attribute thirteen of the SYSTEM file item named POINTER-FILE. In SET.MODULOS, the separation is always 1, but lines 36 to 41 can be replaced with any algorithm for computing modulos and separations.

Once attribute thirteen has been adjusted, lines 45 to 55 present the results and ask the operator for approval to file. This is convenient for debugging and for giving the user a chance to override the program. Often, a new modulo may be calculated but should not be used, such as when a scratch file happens to be empty (implying the file should have a small modulo), but the file really needs a larger modulo because the operator happens to know the file will be filled with data later. In SET.MODULOS, line 56 only WRITEs out the file definition item containing the new modulo if the operator approves, but this can easily be changed to always update the modulo, or to use some other condition. Finally, line 58 goes back to repeat the whole process for the next STAT-FILE item.

The SET.MODULOS program is simple, clean code that is not only a useful tool, but also provides an excellent vehicle for any modulo adjustment algorithm that an installation chooses to use.

Some New Subscribers

Judie Rands
Molectron Corp.
Sunnyvale, CA

Dan W. Olds
Wofford College
Spartanburg, SC

Bill Vaughn
Wildish Sand & Gravel Co.
Eugene, OR

Jim Leggett
Shaw Oxygen Co., Inc.
Monroe, LA

Black Box Formatting

A procedure for emergency formatting of disk drives with black box controllers is described.

Occasionally, an installation's disk drive hardware will break down. If the failure is serious enough, all kinds of symptoms can suddenly begin appearing: ampersands can start filling display screens, files can develop group format errors and lose data, the system can crash and hang.

If data has indeed been lost because of damage to files, then often the only recourse is to restore the data or even the whole file system from backup tapes. Unfortunately, some disk crashes can be so bad that it is necessary for the disk to be reformatted before the system can begin writing data to disk again.

Before Microdata's 4.1 release of the operating system, the normal procedure for reformatting a disk first required the arrival of the customer engineer and the engineer's disk diagnostic tape, which contained the disk formatting program. (Once booted, 4.1 tapes now offer the "D" option to allow disk maintenance.) But many Microdata systems, especially the older units, have a so-called "black box" controller instead of the newer, preferred "single board" controller. Although it is not well known, it is true that the black box is capable of formatting a disk.

If a disk with a black box controller has crashed, and if there is no need to retain the contents of the disk any longer, and the disk needs immediate reformatting but a diagnostic tape is unavailable, try the following procedure:

1. Halt the system.
2. Open the small hinged panel at the rear of the black box controller.
3. Observe the eight toggle switches and the eight red lamps that are now exposed. All switches should be found in the down position, and only the leftmost red lamp should be lit.
4. Starting with the rightmost toggle switch, set the eight switches to the following positions: U D U D U D D U where U indicates up and D indicates down. The red lamp will turn off.
5. Once the disk has been formatted, only the leftmost red lamp will turn back on.
6. Starting with the leftmost toggle switch, depress each switch to the down position.

If any other pattern appears in the lights at the end of Step 5 above, an error has occurred and the procedure should be repeated.

This procedure has been used successfully at least once after a disk crash to reformat a 50 megabyte Reflex drive to allow an emergency file restore, pending the arrival of a disk diagnostic tape containing the standard formatting program. Since the black box controller does not assign alternate tracks when a disk is formatted using these steps, the procedure should not be used unless absolutely necessary.

utilities

Utility programs have saved many a programmer from the less appealing aspects of software development and maintenance. Reformatting code, stripping files clean of control characters, converting data from one format to another — all are examples of tasks best delegated to a program and not a programmer.

Good programmers will collect a "toolbox" of utility software to use from time to time. If you have a useful utility, send it in for publication. A regular feature in Pragma will be this Utilities Department, where good software tools will be spotlighted.

TRIM.DELIM and PROFILE

A program for removing superfluous system delimiters in file items is presented. The basic design of the program is expanded to create a second utility that helps analyze the structure and content of undocumented files.

Attribute marks, value marks and subvalue marks are the delimiters that the system software uses to separate pieces of data in file items. When an item is output to the screen or printer with the COPY verb, the delimiters appear as carriage returns, right brackets and back slashes. But just because a file item contains many attribute, value and subvalue marks doesn't necessarily mean that all of those delimiters have to be stored in the item. Since a value of null is always assumed for a non-existent piece of data, the position of null data is always implied, and in many cases null values do not need to be explicitly identified via system delimiters. This means that data items often contain trailing delimiters that can be trimmed away to save storage space and improve file access times.

TRIM.DELIM is a program for trimming away superfluous system delimiters, and is listed in Figure 1 on page 12. Figure 2 shows a typical data item before and after being processed by the TRIM.DELIM program: the data item is reduced from 19 attributes and 119 characters to 17 attributes and 73 characters, but all data remains untouched and unchanged as far as any programs that access the data might be concerned. This one item has been shrunk by 39%, yet it will continue to work just as before for any software that reads or writes the item. If every item in the same file has a similar, nearly equivalent excess of unnecessary delimiters, then the whole file can be reduced by 39% simply by processing every item in the file with the TRIM.DELIM program.

Not every file on a system will be full of such items, but it is not unusual to find at least one or two files with items carrying extra delimiters. A good approach for identifying TRIM.DELIM candidates is to first sort the STAT-FILE by descending frames used, to get a list of all files, biggest first. Then go through the list, from biggest file on down, to determine the location (account) and name of each candidate file. While logged on to the appropriate account, use the COPY verb to browse through a few items in each file to see if any items exhibit excess delimiters trailing after non-null values like the "before" example in Figure 2. Files with many multivalued attributes that are written by DATA/BASIC programs will often show the symptoms of too many delimiters. If a file with such items is found, and there is a significant number of items in the file, then substantial savings on file space can be realized by using TRIM.DELIM to shrink each file item. Smaller files with only a few items are generally not worth the effort.

The listing of TRIM.DELIM presented here is a simple version that processes only one item at a time. The program prompts for the name of a file, and for the name of an item in the file. The existing item is retrieved, and the program then

```
    TRIM.DELIM
001 PRINT "FILE":
002 INPUT FILE
003 OPEN FILE ELSE STOP
004 PRINT "ID":
005 INPUT ID
006 READ ITEM FROM ID ELSE STOP
007 NEW.ITEM = ""
008 AM = COUNT(ITEM,CHAR(254))+1
009 FOR I = 1 TO AM
010   VM = COUNT
      (ITEM<I>,CHAR(253))+1
011   FOR J = 1 TO VM
012     SM = COUNT
        (ITEM<I,J>,CHAR(252))+1
013     FOR K = 1 TO SM
014       VALUE = ITEM<I,J,K>
015       IF VALUE # "" THEN
          NEW.ITEM<I,J,K> = VALUE
016     NEXT K
017   NEXT J
018 NEXT I
019 WRITE NEW.ITEM ON ID
020 STOP
021 END
```

Figure 1: The TRIM.DELIM program.

```
001 540]510]234]  |  001 540]510]234
002 ]]]3]]]2]]]   |  002 ]]]3]]]2
003 23\33\\\]]    |  003 23\33
004 ]]            |  004
005 ]]            |  005
006 ]]            |  006
007 ]]]]          |  007
008 ]]]]3         |  008 ]]]]3
009 ]]]]34        |  009 ]]]]34
010 ]]]]          |  010
011 41]74]57]]    |  011 41]74]57
012               |  012
013 477]768]34]]  |  013 477]768]34
014 ]]]]]]]]]     |  014
015               |  015
016               |  016
017 32]3]\\       |  017 32]3
018 ]]\\          |
019               |
```

Figure 2: A data item before (left) and after (right) processing.

proceeds to create an equivalent copy of the item by only duplicating non-null values. Lines 9 to 18 search through each attribute in the item. For each attribute found, lines 11 to 17 search through each value in the attribute. For each value found, lines 13 to 16 search through each subvalue in the value. Line 15 finally copies the resulting, extracted piece of data to the new item if it is not null.

Once the effect of TRIM.DELIM has been demonstrated and used with confidence, it is usually desirable to modify the program to automatically trim every item in a file, instead of just one item. This can be accomplished by deleting line 4, changing line 5 to a READNEXT ID ELSE STOP and inserting a GOTO (READNEXT) after line 19.

Although TRIM.DELIM often runs slowly because of the large number of nested FOR loops that can end up being executed, one pass through a file is usually all the trimming that is needed for some time, because the turnover of items in large files, as a percentage of all items, is often fairly low.

Besides superfluous delimiters, another problem related to the contents of files is that complete documentation describing a file's internals may be lacking or unavailable. It is not unusual for a programmer to be given the task of writing a program to create, change or use the data in an existing file for which there is no documentation or other source of information describing the content and structure of the file. When faced with such a task, it is sometimes helpful to have a program that scans the existing items in a file and then reports what has been found, to serve as the basis for more detailed and comprehensive documentation, and so that the programmer will have a better feel for just what data is going to be manipulated.

PROFILE is such a program for scanning and reporting on the contents of files, and is listed on page 13. PROFILE scans every item in a file and outputs a report of six columns showing:

(1) AMC, the attribute mark count, or simply the attribute number, of the attributes found to have data, where "data" means any character, even a value or subvalue mark.

(2) FOUND, a count of the number of such non-null attributes found.

(3) MIN VAL, the minimum value (which may have been stored as an attribute, value or subvalue) in the attribute.

(4) MIN ID, the identifier of the item containing the minimum value.

(5) MAX VAL, the maximum value (which may have been stored as an attribute, value or subvalue) in the attribute.

(6) MAX ID, the identifier of the item containing the maximum value.

The first row in the report is for AMCs equal to zero (the item identifier), so FOUND for the first row is a count of the file's items (including any null item identifier). Since an item's attribute zero and an item's identifier are the same thing, MIN and MAX VAL are the same as MIN and MAX ID for the first row, so the program doesn't bother posting any value under MIN VAL or MAX VAL in that case.

Continued on page 43

```
001 PRINT "DATA FILE":
002 INPUT FILE.NAME
003 OPEN FILE.NAME ELSE STOP
004 PRINT "REFRESH WITH EACH ITEM":
005 INPUT REFRESH
006 REFRESH = (REFRESH="Y") OR (REFRESH="YES")
007 FOUND="" ; MIN.VAL="" ; MIN.ID="" ; MAX.VAL="" ; MAX.ID=""
008 MAX.AMC = 0
009 SELECT
010 PRINT CHAR(12):
011 100 *
012 READNEXT ID ELSE
013    IF NOT(REFRESH) THEN GOSUB 200
014    STOP
015    END
016 FOUND<1> = FOUND<1>+1
017 IF (ID<MIN.ID<1>) OR (MIN.ID<1>="") THEN MIN.ID<1> = ID
018 IF ID > MAX.ID<1> THEN MAX.ID<1> = ID
019 READ ITEM FROM ID ELSE GO TO 100
020 AM = COUNT(ITEM,CHAR(254))+1
021 IF AM > MAX.AMC THEN MAX.AMC = AM
022 FOR I = 1 TO AM
023    A.TXT = ITEM<I>
024    COL.PTR = I+1
025    IF A.TXT # "" THEN FOUND<COL.PTR> = FOUND<COL.PTR>+1
026    VM = COUNT(A.TXT,CHAR(253))+1
027    FOR J = 1 TO VM
028       V.TXT = A.TXT<1,J>
029       SM = COUNT(V.TXT,CHAR(252))+1
030       FOR K = 1 TO SM
031          VAL = V.TXT<1,1,K>
032          IF VAL # "" THEN
033             IF (MIN.VAL<COL.PTR>="") OR (VAL<MIN.VAL<COL.PTR>) THEN
034                MIN.VAL<COL.PTR> = VAL
035                MIN.ID<COL.PTR> = ID
036                END
037             IF VAL > MAX.VAL<COL.PTR> THEN
038                MAX.VAL<COL.PTR> = VAL
039                MAX.ID<COL.PTR> = ID
040                END
041             END
042       NEXT K
043    NEXT J
044 NEXT I
045 IF REFRESH THEN GOSUB 200
046 GO TO 100
047 *
048 200 *
049 PRINT @(0,0):
050 PRINT "AMC FOUND.... MIN.VAL.. MIN.ID... MAX.VAL.. MAX.ID..."
051 PRINT
052 FOR I = 0 TO MAX.AMC
053    COL.PTR = I+1
054    NUM.FOUND = FOUND<COL.PTR>
055    IF NUM.FOUND > 0 THEN
056       PRINT STR("0",3-LEN(I)):I:
057       PRINT NUM.FOUND "R#10":
058       PRINT MIN.VAL<COL.PTR> "R#10":
059       PRINT MIN.ID<COL.PTR> "R#10":
060       PRINT MAX.VAL<COL.PTR> "R#10":
061       PRINT MAX.ID<COL.PTR> "R#10"
062       END
063 NEXT COL.PTR
064 RETURN
065 *
066 END
```

PROFILE
PROGRAM LISTING

SYSMAP

Part 2: File Format Input

This second article in a series on the use of cross-references presents the SYSMAP input program for maintaining the FF file. Also, various examples of code are given that demonstrate the need for a cross-reference system.

In the last installment, *SYSMAP Part 1: A Cross-Reference System* (Pragma #1, page 22), the six SYSMAP files were introduced. Of those six, the FF file serves as SYSMAP's data dictionary, and is designed to contain the minimum information necessary to explain the contents of any file being cross-referenced by SYSMAP. Specifically, the data maintained in the FF file names all attributes in all files, indicates the number and structure (attribute, value or subvalue) of each attribute, and includes a description of what each file attribute is used for.

The listings on the next page form a program for creating and changing the data in SYSMAP's FF file. By actually using this program, the descriptions of the FF file itself and the other five SYSMAP files were entered and then output with ENGLISH to generate the listing of all SYSMAP files that was included in *Part 1.* In the same way that this program can be used to input descriptions of SYSMAP's files, the program should be used to input the descriptions of any other files that an installation is interested in cross-referencing, such as payroll files, invoice files, inventory files and so on. Once all file descriptions have been input and established, other SYSMAP programs can be used to record the relationships between all files and programs.

Part 1 identified the different types of program and data interactions that can occur, such as procs invoking programs, programs reading files, and file attributes translating other file attributes. But exactly in what form do these relations appear? Here are some actual code fragments demonstrating the types of interactions that must be identified and recorded with SYSMAP to insure a fully documented and cross-referenced software system.

Procs Invoking Programs. The following proc first invokes an uncataloged program named INPUT and then executes a cataloged one named OUTPUT:

```
PQN
HRUN BP INPUT
P
HOUTPUT
P
```

All such program references by procs must be recorded in SYSMAP's XB file. If the code in any program is going to be modified, the programmer must first check the XB file to find which procs invoke the program, thereby determining which procs may be affected by the program change.

Procs Invoking Procs. The following proc first calls a proc named MENU and then transfers control to a proc named QUIT:

```
PQN
[PROC.LIB MENU]
(PROC.LIB QUIT)
```

All such proc references by procs must be recorded in SYSMAP's XP file. If the code in any proc is going to be modified, the programmer must first check the XP file to find which other procs invoke the proc, since those procs may also be affected by the change.

Procs Referencing Files and Dictionary Words. The following proc accesses a purchase order file and uses three of the file's dictionary words:

```
PQN
HSORT PO BY VENDOR.NUM
H VENDOR.NUM VENDOR.NAME PO.DATE
P
```

All such file references must be recorded in SYSMAP's XD file. If the specifications in any dictionary word are going to be modified, the programmer must first check the XD file to find which procs reference the word, thereby determining which procs may be affected by the dictionary word change.

Dictionary Words Referencing Dictionary Words. The following dictionary word invokes two other words:

```
     STD.TOTAL
001 A
002 0
007 MD2
008 A;N(STD.LABOR) + N(STD.MATL)
009 R
010 10
```

All such word references must be recorded in SYSMAP's XD file. If the specifications in any dictionary word are going to be modified, the programmer must first check the XD file to find which other words invoke the word, since those words may also be affected by the change.

Programs Referencing Files. The following program fragment reads two attributes (WANTED and SHIPPED) and writes a third (DUE):

```
READU ORDER FROM SO.NUM ELSE STOP
ORDER<DUE> = ORDER<WANTED> - ORDER<SHIPPED>
WRITE ORDER ON SO.NUM
```

All such file references must be recorded in SYSMAP's XF file. If the use of any file attribute is going to be modified, the programmer must first check the XF file to find which programs read or write the attribute, thereby determining which

programs may be affected by the change in the attribute.

Dictionary Words Referencing Files. The following dictionary word adds two attributes of a file:

```
     SUM
001 A
002 0
008 F;3;5; +
009 R
010 10
```

All such attribute references must be recorded in SYSMAP's XF file. If the use of any file attribute is going to be modified, the programmer must first check the XF file to find which dictionary words read the attribute, thereby determining which dictionary words may be affected by the change in the attribute.

Translate Conversions Referencing Files. Translate conversions can reference file attributes from a number of different locations.

From a program: PRINT OCONV(PART.NUM,"TPARTS;X;;1")
From a proc: IBH%1:TPARTS;X;;1:
From a screen: TPARTS;X;;1 or (VPARTS)
From a dictionary: TPARTS;X;;1

All such file references must be recorded in SYSMAP's XT file. (The XT file is also used to record the rather rare cases of file input/output by procs using F-WRITE and similar commands.) If the use of any file attribute is going to be modified, the programmer must first check the XT file to find which programs, procs, screens or dictionary words translate the attribute, thereby determining which of them may be affected by the change in the attribute.

In the next installment, the remaining programs for maintaining SYSMAP file input will be presented. Ⓟ

```
01 OPEN "DF" ELSE STOP "SYSMAP1"
02 READ SCREEN FROM "#GET.FF" ELSE STOP "SYSMAP2"
03 OPEN "FF" ELSE STOP "SYSMAP3"
04 PREV.FILE.ATR = ""
05 100 PRINT CHAR(12):
06 200 INPUT ITEM USING SCREEN,PREV.FILE.ATR SETTING NEXT.STEP ELSE STOP
07 FILE.ATR = ITEM<1>
08 PREV.FILE.ATR = FILE.ATR
09 IF FILE.ATR = "" THEN GO TO 200
10 ATR = ITEM<2>
11 IF ATR # "" THEN FILE.ATR = FILE.ATR:"*":ATR
12 READ ITEM FROM FILE.ATR ELSE ITEM = ""
13 INPUT ITEM USING SCREEN, ITEM AT NEXT.STEP ELSE GO TO 100
14 WRITE ITEM ON FILE.ATR
15 GO TO 100
16 END
```

```
01
02
03 FILE]ATR]AMC]AVS]DESC]OK]BLNK22
04
05 Create or change file format data\\\
   \\\\                    File:\\
       Attribute:\\                    AM
   C:\\                     AVS:\\
     Description:
06 0,0
07 ]]AVS\DESC]]BLNK22
08
09 S]S]S]S]V]D]D
10
11 L]L]L]L]L
12 1]2]1]2]3
13 Y]]]]]]Y
14
15
16 7,23]9,23]11,23]13,23]15,23]]22,0
17 20]20]2]2]45]]1
18 ]]]]]]@
19
20 20]20]2]2]45]]1
21 ]]]]]]L
22 File name:]Attribute name:]AMC:]AVS:
   ]Description:]File (FI) or exit (EX)
   ?
23 22,0]22,0]22,0]22,0]22,0]22,0
24 22,10]22,15]22,4]22,4]22,12]22,23
25 70]65]76]76]45]57
26 ]]]].
27 20]20]2]2]45]2
28 ]]](‘A’)0(‘V’)0(‘S’)
29
30
31 ]]MDO
32 ]F]]]](‘FI’)F\GOK
33
34
35
36
37
38 ]]]]DESC
39
40 (‘EX’)E
```

Ⓟ

The program that inputs the SYSMAP FF file.

user profile

Are all data processing installations the same? How do managers actually manage, how do programmers program, how do operators operate, how do users use their data processing systems? What defines the leading edge of current, modern information processing?

In this regular department, Pragma will be interviewing personnel at a variety of installations, to reveal the who, what, when, where, why and how of actual data processing organizations.

Galinski, Hamburg & Company, a small CPA firm of just over 30 employees, is one of Pragma's Pennsylvania subscribers. Located in Hatboro, about half an hour out of Philadelphia, the company offers accounting and data processing services to its clients. For this issue's user profile, Pragma interviewed Joe Kempter, a CPA and the Galinski Hamburg partner responsible for data processing.

Pragma: How did Galinski Hamburg initially get involved with computers?

Kempter: What we did was define our market, and right from the start we wanted to have the capability where our clients can have all the benefits of a computer without all the hidden costs, such as the cost of personnel and the cost of managing that function. Immediately our goal was to go to a time-sharing type of environment where clients have their own terminals, their own printers, and utilize our hardware plus the people that we have at Galinski Hamburg.

Pragma: When did you acquire a system?

Kempter: When I first came on board about five or six years ago, about 1976. At that point we looked at various vendors. IBM, Microdata, Basic Four. I think it got down to three. Then we decided on the Microdata system. What we got was a very small system, as I reflect upon it. It had 32K core memory, a 10 megabyte disk drive, and a 165 character per second Scribe printer. We utilized that for our processing, plus we got a software package from SMI out in Chicago which was called their Business Control Package. It had accounts receivable, accounts payable, payroll, general ledger and financial statements. That was our starting point. We stayed with that system for a period of about nine months, and then we started to run out of capacity. We added a second 10 megabyte disk drive. Now as I think about it, the biggest mistake we made there was that we didn't get enough hardware to begin with. We really should have bought a lot more.

Pragma: How was it determined what your initial configuration should be?

Kempter: We looked at what we were going to do and I think we were a little bit pessimistic. The thing really took off. Before you knew it, we ran out of mass storage and we always had to go into what I call an offline processing mode, where we had to take certain clients and transfer them off to tape, then bring them back in again. It got pretty horrendous. We made a serious error in not really evaluating the market as well as we should have. We were too conservative, I guess, being accountants.

Pragma: Was cost a problem?

Kempter: No, it wasn't really a cost matter. The cost of hardware was really cheap compared to the cost of labor that we

got involved with, in operating in this offline mode. We were continually growing. We were getting heavily involved in customized programming for our clients. We talked to Microdata and we told them what our goals and objectives were. Microdata told us that probably the best route to go was to get two computers. What they suggested we do is take our system that we had, a 20 megabyte system which had 64K of memory, and utilize that in a software development mode. And they suggested that we get a Microdata 6000 series with a 50 megabyte disk drive, and I think it had 128K.

Pragma: This was about a year after your initial acquisition?
Kempter: I think it was about two years.

Pragma: So two years after you got your first 10 megabyte system, you were converting into two separate machines?
Kempter: Yes. Our plans were that one machine would be used for software development and the other machine would be used for our timesharing services. That worked out pretty good. Once again, we grew faster than we had planned, which was sort of a nice problem to have. We were growing to such an extent that the timesharing system got very loaded, and the clients weren't getting the type of response that I thought they should. We knew we had serious capacity problems. We couldn't fit any additional clients on the system. We had to do something, so once again we went back to Microdata, about a year, a year and a half ago. They suggested that we go to one system, which was a Sequel. They felt is should be able to handle both our software development work and our timesharing services, and to take care of our needs with expansion capabilities over the next three to four year time period. One thing I learned is buy more than you need as far as hardware. The cost of hardware is really cheap compared to the cost of the people.

Pragma: Did you shop around?
Kempter: We talked to several other computer vendors, not in any great detail. We looked at the ADDS Mentor system, we talked to the people from Ultimate, we looked at Prime. But then we saw the darn conversion effort that was involved to convert our programs over to operate on the Ultimate or the ADDS Mentor or Prime. The easiest route to go was to stick with Microdata and go with the Sequel.

Pragma: How was the conversion effort and costs for going with those other vendors calculated?
Kempter: Well, they told us it would only take a couple weeks, and we didn't believe them, because all our programs are written in PROC. They're not in DATA/BASIC. Even though these other vendors told us that the conversion would be minimal, we did not believe them. I don't think they realized how many programs we had to convert. Thousands of programs. Not hundreds, but thousands.

Pragma: There wasn't any attempt by these vendors to refer you to satisfied customers that could give glowing testimonials about easy conversions?
Kempter: No, not really.

Pragma: How is it that so much of your software is done only in PROC? I would think that for many kinds of applications, especially financial, using only PROC would be a big limitation.
Kempter: Right. When we got our initial system, we had purchased a package from SMI. Everything was written in PROC. Then we made a mistake. When our data processing manager came on board, the only language he knew at that point was BASIC, but then he had to learn PROC. So he learned PROC and liked it so much that he insisted on using it for further applications. We got to the point where we saw that that was a bad decision, so we made a policy that no further applications would be written in PROC. But we already had the packages that were purchased, plus what we had done internally. It was a mistake on our part, but nothing in the future will be written in PROC. Now all our programming is in DATA/BASIC.

Pragma: What happened after you decided on Sequel?
Kempter: Oh, they were promising us delivery in January. The delivery date was constantly postponed.

Pragma: When did the promising start?
Kempter: Back in November. Then we got it in May or June.

Pragma: How was the conversion?
Kempter: Oh, simplistic. We really did a lot of homework before we even got the system. We had several concerns about the Sequel. The first concern was, would our foreign equipment operate with the Sequel? Communications devices or printers or terminals. Naturally, Microdata said yes it would, but we didn't believe them. So, before our system was installed, we took all our foreign equipment up to Union, New Jersey to their branch office and we did a test. The second question we had was, will our software work on the system? They said yes and once again we didn't believe them. So once again we went to Union, we tested it, and it worked beautifully. The third question, probably the most important question of all was, what would the response time be? They told us it would be fantastic. We said we really don't believe you, we want to go up and see it. So we went up to Union and we went through various benchmark tests on the Sequel. I think we had 64 terminals going, trying to simulate our environment: people doing data entry, our programmers doing compiles on their programs, clients processing reports. We tried to really simulate that environment up at Union. And we did. I think we had about 10 terminals doing compiles, others doing sort selects on some very large files, other ones processing reports. The response time was very good. It slowed down, naturally, as we went to each succeeding application we were trying to do. But even with everything going at one time, response time was there. It was an unbelievable improvement over what we had before.

Pragma: When was the period of time all this homework was taking place?
Kempter: We were doing that from as soon as we signed the contract, January 20th, through May.

Pragma: How come you didn't test before you signed the contract?
Kempter: There weren't any Sequel systems available. In our contract we had provisions that, if certain things weren't accomplished, we did not have to accept the system. I'd tell anybody who gets into buying a large system like this that the first thing they should do is find themselves legal counsel that has experience in the computer area. We found an excellent firm that has a person — all he does is deal in computer type contracts. He really covered us in every way possible. He drew up an excellent contract.

Pragma: How did he cover you on the late delivery?
Kempter: Since they were late, they had to give us additional interim hardware. They had to give us an additional 50 megabyte disk drive. Then, if they were late beyond that, he had additional penalties built into the contract and what have

you. He really did an excellent job.

Pragma: What were the events once the Sequel arrived?

Kempter: That was really simple. It came in, it was certified by the customer engineers. They went through their testing of the system for three or four days, then they turned it over to us. We operated three systems at one time. We would gradually take one client from the 50 megabyte system, and transfer him over to the Sequel and make sure he was running OK. Gradually, we got all the clients off the 50 megabyte system, and up and running on the Sequel. This took a period of a month or two. We didn't do it all in one fell swoop.

Pragma: How much testing of the system did you do before you actually moved a client over for the first time?

Kempter: All our testing really was done when we were up in Union at their branch office. We tested the majority of our programs there, so we didn't do too much testing. But we monitored the clients who did transfer over, very closely. What we transferred over first was all our batch work, because that was easiest for us to control. If we saw any problems there, we could correct those and the clients would never know about it. It would be transparent to them. So, those were the first applications that we transferred over. Then we saw if we had any problems. Such as, we went to the Printronix printer and before we had a Dataproducts. A Printronix only accepts 132 characters. We had various reports printing out that were beyond 132 characters. So we had to make program modifications like that. But nothing really substantial.

Pragma: How does the Sequel meet your original expectations?

Kempter: I'd say it meets probably 60% right now. We're just experiencing a few problems which Microdata is working on. One has to do with the amount of data you can keep in memory. Since the memory is so large, you have all these records that could be in memory that have been updated and not yet written out to disk. Now if anything happens, if the system should go down for any reason, and you have to do a coldstart, all that data that was in memory that hasn't been updated is lost. There's something that Microdata came up with to determine how long the data is going to stay in memory. You can set a parameter in the system. Initially, a system comes with this parameter set at two billion something. They made us take it down to four, but it really slows down your response time. We complained about that, and they had us set it back up to one billion. What they're saying is keep very little data in memory. As soon as it's updated, transfer it out to disk and therefore, if you do go down, that data won't be lost.

Pragma: Have you had much problems with down time?

Kempter: No, not really, when you look at the total time period. It's been installed since May and we've had about two weeks of major problems.

Pragma: Have you actually experienced an outage where you lost something because that parameter was not low enough?

Kempter: Yes. What that does is get the clients upset. Microdata promised us they will get this problem resolved. It is better now that the parameter is back at one billion. At four, it was really slow.

Pragma: How much of Galinski Hamburg's success with computers contributed to the company's growth?

Kempter: We think it has enabled us to grow because we present a more progressive type image to our clients. That we are a CPA firm up to date as far as technology is concerned, and we're trying to provide the clients with the best services that are possible. We think we're more progressive than most small local CPA firms. Matter of fact, we think we're even ahead of the Big 8 accounting firms. We have gone to some of their demos and what they're doing, and either we're doing the same thing or we're a step ahead of them. We try to present a very progressive image to our clients, plus we firmly believe that our clients — small closely held corporations — should not own their own computers.

Pragma: Why is that?

Kempter: They don't have the time or the expertise to manage them effectively. It's very hard for them to get used to the discipline and the controls that a data processing environment imposes on you. To go that route immediately from day one is very difficult. They're better off going with our approach, a building block approach, a modular approach, whereby a client starts off very slowly in the batch mode. He gets used to the discipline, the control. People are trained gradually. He determines how he can utilize that system to give him the information he needs to make his decisions. We have seen too many disasters where a client goes out there and day one gets a computer, brings it in house, no one's trained, and they have a big disaster on their hands. We're saying yes, maybe go that route but do it gradually over a two or three or four year period. We have done that with some clients who have started off with us in a batch type environment and right now they have their own Microdata systems.

Pragma: But Galinski Hamburg essentially got their own computer and did their own in house effort and succeeded. Why is it that you were successful and you think some of your clients couldn't be?

Kempter: Because we really have excellent people. I think that the reason we have been so successful is the fact that we have found really good people and have kept those people with us as a team over a long period of time. We have been growing and our turnover is really nonexistent. At the time I came on board, we also hired a data processing manager who had about eight years experience. We have three programmers who have been here on the average of, I'd say, three years, and we just

Some New Subscribers

Mike Blue
A & A Glass & Mirror
Miami, FL

Paul T. Craven
Highland Import Corp.
Marlboro, MA

Alan M. Thompson
Hudson Oxygen Therapy Sales Co.
Temecula, CA

Lynn Anderson
Medical Computer Billing Service Inc.
Augusta, GA

don't experience any turnover in the data processing area.

Pragma: Why is that?

Kempter: One reason is that we reward our people very well. We have a good work environment. I try to keep them involved in all the decision making processes that we go through, utilize basically a team approach. We have a lot of unique fringe benefits that we offer our employees. We have a retirement plan where we take 10% of their salary and put it away each year in a Keogh type investment vehicle. In the summer months, in June, July, August and September, we operate on a four day work week basis, so half our data processing staff is off on Friday, half is off on Monday. We invest heavily in their continuing professional education. We're constantly sending them to courses either at Microdata or various other locations to try to better themselves. All the programmers have to take at least an introductory accounting course. I think we invest a lot of dollars in them, and I think we treat them very good. I think some of the unique things that we do, like the four day work week, and making them a key part of the decision process where they really are involved, is the reason they stay with us, because it's kind of unusual. We have more turnover on the accounting side than we do on the data processing side.

Pragma: What are the applications you provide for your clients?

Kempter: The basic accounting applications, what we call our bread and butter. That would be accounts receivable, accounts payable, general ledger and financial statements and payroll. Then we get into other applications. Order entry, invoicing, sales analysis, job costing, quotations, financial modeling, labor reporting, perpetual inventory, record keeping. We have an awful lot in the job cost area.

Pragma: Do you do any auditing of client's work that is based on your own services?

Kempter: Yes, we have several clients where we do their computer processing as well as do the audits.

Pragma: Isn't that some sort of conflict of interest? How can you audit something that is based on what you yourself created?

Kempter: Our person who is doing the audit is in a different department of the firm than the person who is providing the computerized service. We can even do a better job because of the fact that our auditors are more familiar with data processing and what a computer can do and what it can't do. Other people out there who are auditors do not have any experience in this area.

Pragma: What's an example of a mistake your data processing people might make when dealing with a customer?

Kempter: At times they might not speak the customer's language. They might be a little too abrupt, they might think that the customer is at the same level of knowledge as they are. At times I think they're not patient enough. I guess the biggest thing is patience and being able to establish a rapport with the clients, to loosen them up a little bit, to have a sense of humor. You have to have an awful lot of people skills to keep the clients happy, to take care of their problems and do it in a very diplomatic fashion.

Pragma: What would you guess your data processing people would like to see change at Galinski Hamburg?

Kempter: I think that what they would like to change most is to better define our systems before we actually get into programming them. I don't know if this is even possible with the type of clients that we deal with. We're dealing mostly with small corporations. They range in size from a million to five million in sales. And most times the people we're dealing with really don't know what they want. A lot of systems are designed and programmed on a trial and error type basis. We tried to go to a more sophisticated type approach, where we went into methodology like I used when I worked for a bigger corporation, at Johnson and Johnson. There we went in and we determined what a client's requirements were. We laid out all the screen layouts, the report layouts, the files. We had the client sign off. We completed the programming, they started to use it and they said no, that's not right. Could you make a change here, could you make a change there? This sort of drives the analysts and the programmers crazy. We've talked an awful lot about this, my data processing manager and myself, and I don't know if we'll ever be able to approach this any differently, with the type of clients we're dealing with, who are engineers or salesmen who have started their own business and really don't have the time to spend with you to really determine what they want. The way they determine what they want is by utilizing some system over a period of time.

Pragma: What internal needs do you use your system for?

Kempter: We use it for what we call our client time and billing system, whereby we keep track of all the time that each individual spends with a client, accumulate that by client and then use that for our billing process and to evaluate the productivity of our employees and to evaluate the productivity of the various types of services we are offering to our clients. The second way we utilize it is to schedule the various audits and tax work that has to be done within the firm. We also developed the 1040 system. Not to prepare 1040s, but to really track the 1040s from the time they come in to our firm until the time they're delivered to the clients. We know at any point in time exactly what the status of a 1040 is.

Pragma: How well do you feel Galinski Hamburg is automated internally as oppposed to how well it has automated some of its customers?

Kempter: I think that we have automated almost everything

Continued on page 30

Pragma is Preparing an Index

Pragma will be periodically publishing a complete master index for all articles appearing in **Pragma**, as a reference tool for its readers. Articles will be cross-referenced by keywords found in each article's title, by the author's last name (for submittals), by additional miscellaneous keywords for various subject areas, and by department. Also, all advertisements will be indexed by vendor, product name and product type.

benchmarks

Are you thinking of doing an upgrade to your hardware or software? Are you comparing throughput and performance while shopping around for a system? Have you converted from one machine to another? Be sure to send in your benchmark statistics to Pragma, so the results can be featured in this department and shared with other installations.

LOOP vs. LOCATE

An investigation to determine which is faster.

It seems that programmers either love the LOCATE statement or they hate it. LOCATE supporters love the power and flexibility available in DATA/BASIC's most esoteric statement: the concise manner in which complicated searches can be stated, the ability to deal with sorted lists, and the elegant handling of search failures possible with the ELSE clause. LOCATE detractors hate the strange and confusing statement. They hate the unmemorizable syntax (which changed with 3.2), the obscure search rules (especially when looking for nulls), and the 3.2B bug that made the whole thing unreliable.

But how much of a difference to the computer does a LOCATE make? If one program uses a LOCATE, and an equivalent program does not, which is faster? The rather interesting results are summarized in Figure 1, below. (Thanks go to Pragma subscribers Davin Lawson and Kent Sulprisio at Zentec Corporation in Santa Clara, California for help in gathering many of the statistics.)

In order to create a benchmark that would heavily exercise a LOCATE statement (or its equivalent), the following "nonsense" task was devised:

1. Read a data item (see Figure 2).
2. Search each attribute and find the attribute equal to only "Z".
3. For all lower attributes, search each value and find the value equal to only "Z".
4. For all lower values in each lower attribute, search each subvalue and find the subvalue equal to only "Z".
5. Repeat the whole task one hundred times.

Figure 3 lists the two programs used for the timings. Each is designed to perform the given search task and report the elapsed time in seconds. One version avoids the LOCATE statement and uses LOOP statements instead. The other program relies only on LOCATE statements. Note that the programs do not assume or support any kind of sorted data within the search item. (In other words, there is no BY clause in the LOCATE.) Also note that the LOCATE version can detect a search failure because of the ELSE clause, while the LOOP version is coded to assume data will always be found. Line 14 in each program is simply extra insurance that the search is successfully programmed and will work.

The results indicate that the LOCATE version is roughly three times faster than its equivalent code. It is also interesting to notice the speedup allowed by MOS compared to core, when not on a Sequel. Because of constraints on machine availability, the MOS and Sequel timings were obtained while other users were logged on, but not very active. Even so, the results show a decisive difference between LOOPs and LOCATEs. The core benchmark ran on a completely idle machine.

Although a true and complete measure of software "efficiency" must take into account factors such as documentation time, frequency of bugs, difficulty of maintenance, and ease of understanding by programmers, the results presented here might just sway a LOCATE hater to become a LOCATE lover. P

	Elapsed Seconds Core 6000	Elapsed Seconds MOS 6000	Elapsed Seconds Sequel	Ratio Core:MOS	Ratio Core:Sequel	Ratio MOS:Sequel
LOOP Version	265	154	25	2:1	11:1	6:1
LOCATE Version	92	47	9	2:1	10:1	5:1

Figure 1: Elapsed time comparisons, LOOP statement vs. LOCATE statement.

```
    BENCH
001 X\X\X\X\X\Z]X\X\X\X\X\Z]X\X\X\X\X\Z]X\X\X\X\X\Z]X\X\X\X\X\Z]Z
002 X\X\X\X\X\Z]X\X\X\X\X\Z]X\X\X\X\X\Z]X\X\X\X\X\Z]X\X\X\X\X\Z]Z
003 X\X\X\X\X\Z]X\X\X\X\X\Z]X\X\X\X\X\Z]X\X\X\X\X\Z]X\X\X\X\X\Z]Z
004 X\X\X\X\X\Z]X\X\X\X\X\Z]X\X\X\X\X\Z]X\X\X\X\X\Z]X\X\X\X\X\Z]Z
005 X\X\X\X\X\Z]X\X\X\X\X\Z]X\X\X\X\X\Z]X\X\X\X\X\Z]X\X\X\X\X\Z]Z
006 Z
```

Figure 2: Data designed to cause 2,600 searches.

```
    LOOP.BENCH
001 OPEN "MD" ELSE STOP "BENCH1"
002 READ ITEM FROM "BENCH" ELSE STOP "BENCH2"
003 START = TIME()
004 FOR PASS = 1 TO 100
005   A = 1 ; LOOP UNTIL ITEM<A> = "Z" DO A = A+1 REPEAT
006   A = A-1
007   FOR ATR = 1 TO A
008     A.TXT = ITEM<ATR>
009     V = 1 ; LOOP UNTIL A.TXT<1,V> = "Z" DO V = V+1 REPEAT
010     V = V-1
011     FOR VAL = 1 TO V
012       V.TXT = A.TXT<1,VAL>
013       S = 1 ; LOOP UNTIL V.TXT<1,1,S> = "Z" DO S = S+1 REPEAT
014       IF S # 6 THEN STOP "BENCH6"
015     NEXT VAL
016   NEXT ATR
017 NEXT PASS
018 PRINT (TIME()-START):" SECONDS"
019 STOP
020 END
```

```
    LOCATE.BENCH
001 OPEN "MD" ELSE STOP "BENCH1"
002 READ ITEM FROM "BENCH" ELSE STOP "BENCH2"
003 START = TIME()
004 FOR PASS = 1 TO 100
005   LOCATE "Z" IN ITEM,1 SETTING A ELSE STOP "BENCH3"
006   A = A-1
007   FOR ATR = 1 TO A
008     A.TXT = ITEM<ATR>
009     LOCATE "Z" IN A.TXT<1>,1 SETTING V ELSE STOP "BENCH4"
010     V = V-1
011     FOR VAL = 1 TO V
012       V.TXT = A.TXT<1,VAL>
013       LOCATE "Z" IN V.TXT<1,1>,1 SETTING S ELSE STOP "BENCH5"
014       IF S # 6 THEN STOP "BENCH6"
015     NEXT VAL
016   NEXT ATR
017 NEXT PASS
018 PRINT (TIME()-START):" SECONDS"
019 STOP
020 END
```

Figure 3: The LOOP version (497 bytes of object code) and the LOCATE version (462 bytes).

wish list

Users of computer systems should always remember that the nice thing about software (besides the fact that it doesn't break when you drop it) is that programs are relatively easy to change — no soldering gun is necessary. So just because the operating system or the compiler or a system utility happens to work (or fail to work) in some particular fashion now, doesn't mean it has to always be that way. The next time an inspired idea arrives for improving your system, write it down and send it to Pragma for publication in the Wish List. Naturally, all such submittals are eligible under Pragma's author payment program.

The first 23 Wish List items appeared in Pragma #1.

24. User Accessible Passwords. Passwords are the main defense against unauthorized use of an account. But if a user suspects someone knows the password for his or her account, the user must wait for someone with access to the SYSTEM file to change it. Even then, the password can only be changed if trailing items in the SYSTEM file's group are not logged on. Provide an easy way for accounts to change their own passwords. The technique used by the Unix operating system might be a good model, since it seems to be one of the more thought-out password systems.

25. DISPLAY Statement for DATA/BASIC. In order to simplify programs that output to both the display and the printer, provide a DISPLAY statement equivalent to DATA/BASIC's PRINT statement, except that output is always sent to the display screen, regardless of the use of PRINTER ON statements or the (P) option with the RUN verb.

26. SCREENPRO Step Conversions. It is often desirable to specify complicated input/output conversions in SCREENPRO steps. Allow F, A, MF and user conversions to be specified in the step input/output conversion parameters.

27. Editor DE Command Feedback. When any form of the DE command is given in the Editor, there is no feedback at all. This can be confusing, especially since the current line pointer doesn't change. Make the DE command move the line pointer to the line after the last line deleted, and then list the new current line. If too many existing procs have been written that depend on how DE now works, then provide a new name (DL?) for a command with the desired features.

28. File ENGLISH Output. It is often desirable to perform special post-processing on the reports generated by ENGLISH. This could easily be done if ENGLISH output could be sent to a disk file. Unfortunately, output can only go to the printer or tape. Allow ENGLISH commands to send the output to disk files, either as straight text as it might be sent to the printer, or "structured", such as saving each line as a file item, with value marks separating columns, to help simplify data extraction and post-processing by other programs. Allowing DATA/BASIC programs, instead of the LIST formatter, to access the HS area as described under "ENGLISH Interface" in the *Assembly Language Manual*, would be an attractive type of implementation.

29. Contiguously Printed Reports. It is often desirable to spool a message with BLOCK-PRINT and then immediately spool a report, so that the BLOCK-PRINT banner will identify who owns the report and/or where it should be delivered after printing. Similarly, it is often desirable to print multiple reports in succession, or to create reports from multiple ENGLISH commands (such as repeated COUNTs). Unfortunately, output spooled from other ports may end up between the reports which are wanted in succession. Using the various spooler options is cumbersome and requires too much operator interaction. Provide an easy mechanism for users to invoke which will guarantee two or more reports created in succession at a given port will be printed in succession.

30. DATA/BASIC String Manipulation. Changing characters in the middle of a string requires some messy and inefficient code:

```
* REPLACE NTH CHARACTER WITH "*"
STRING = STRING[1,N-1]:"*":STRING[N + 1,LEN(STRING) – N]
```

Instead, allow substring specifiers on the left side of assignment statements, to indicate string replacement:

```
* REPLACE Y CHARACTERS STARTING AT X
STRING [X,Y] = "ABC"
```

31. Limit Multivalue Entry in SCREENPRO. Some applications require limiting the number of multivalues being input. Allow SCREENPRO to limit the number of multivalues that may be entered for a field.

32. Full Page Text Editing. ENGLISH is "page oriented" when reports are displayed, and DATA/BASIC and SCREENPRO allow random cursor addressing so that applications programs can also be "page oriented" when displaying data. That is, the operator gets to see a full page of output and data while reviewing fields or making changes. Unfortunately, the system text editor does not provide such a convenient context for the operator. It is line-at-a-time oriented, as though only a hardcopy terminal was ever being used to edit text. Provide a display oriented text editor that allows the operator to drive a cursor through a page of text, then make changes and have the full page immediately updated (similar to SCREENPRO's painter).

33. E Register Access in PROC. It is possible to test for certain error message identifiers using the IF E command, but there is no way to manipulate the identifier itself (such as using it in a FB command or passing it to a program). Allow the "E register" to be an argument of the MV command, so that the last ERRMSG item identifier invoked can be moved to any buffer. For example, MV %1 E.

34. Subroutine Calls from Correlatives. There are many times when arbitrary custom processing, as coded in a DATA/BASIC program, needs to be invoked from an ENGLISH conversion or correlative. Allow the V/CONV and V/CORR attributes to call DATA/BASIC subroutines. (This facility is already supported on Prime Information.)

35. On-Line System Documentation. Even though vendors produce their documentation via computer, users only have access to printed versions. Fixing typos, reproducing sections of manuals, and creating abridged versions for users must all be done manually. Provide system documentation in machine readable form.

36. DATA/BASIC Runtime Assertions. A good debugging technique is to sprinkle error checks through a program, to make sure that incorrect computations do not take place. Even after the program appears debugged, it's a good idea to leave the tests in the program. Unfortunately, the effort to maintain such checks in the code is often too much trouble. For example,

```
IF X # Y THEN PRINT "LINE 100, X # Y!"; STOP
```

has to be edited each time its line number changes. Provide an ASSERT statement for allowing easy custom error checks. The form ASSERT *expression* would cause program execution to stop (or warn, but continue?) if the expression evaluates to zero. For example,

```
ASSERT BALANCE = (SCHEDULED−DELIVERED)
```

might cause

```
LINE 34 [B777] ASSERTION FAILED!
```

A compile time option to ignore ASSERT statements could be used by those programmers worried about the additional overhead the statements would cause, but as the old story goes, that would be like training with a life jacket and then going on a voyage without one.

37. Control of SCREENPRO Command Mode. It is often too difficult for a program to handle all the possible consequences of an operator using SCREENPRO's command mode. Allow disabling of all or portions of SCREENPRO's command mode by a program.

38. Editor B Command Output. When a B command is given in the Editor, the operator must then give an additional command to find out what the new current line looks like. Make the B command list the new current line.

39. PQ-COMPILE STON Input. For installations that frequently compile procs and always put the object in one file, it is desirable to write a proc to do the compile and automatically supply the PQ-COMPILE verb with the destination file name. Unfortunately, PQ-COMPILE does not take input from the secondary input buffer, so that a proc like

```
    PQ.COMPILE
001 PQN
002 F
003 HPQ-COMPILE SOURCE
004 100 A
005 IF A GO 100
006 STON
007 HOBJECT
008 P
```

cannot be used. Make PQ-COMPILE take destination input from the secondary input buffer, like COPY.

40. PAGE Mode Output. When output is sent to a terminal in PAGE mode, it is often difficult to tell when the page is complete and the system is waiting for a carriage return. Allow a kind of PAGE mode that prompts the user to hit the RETURN key to continue whenever the display is full and there is more output coming. FOOTING would work for ENGLISH except that it causes an unnecessary prompt on the last page. The general prompt capabilities and formatting allowed by FOOTINGs are desirable, however.

41. Undocumented Software. There are numerous undocumented software features, such as the LOGON, LOGOFF and PEEK verbs, and the "R" specification in attribute nine of SYSTEM account definition items that causes an account's logon proc to be invoked after the BREAK key is hit and END is entered. Document all software features.

42. DATA/BASIC LOCATE Syntax. More often than not, the starting position for a LOCATE search is 1, which suggests that a default would be appropriate. Also, there are many times when the SETTING and/or ELSE clauses are unnecessary, so they should be optional too, just like the BY clause. Assume 1 if a starting position is missing in a LOCATE, so that LOCATE "ABC" IN ITEM X means the same as LOCATE "ABC" IN ITEM X, 1. Note that when a starting position is specified, the word "AT" would be more meaningful than a comma. For example, LOCATE "ABC" IN ITEM X AT 5.

43. Editor Up Default. To move up a line in the Editor requires typing "U1". Just typing the more convenient "U" moves up zero lines, a rarely required capability (which can be done anyway by typing "UO"). Let the Editor U command with no argument be the same as the U1 command.

44. PROC PQ and PQN Compatibility. Allow PQ and PQN procs to be called and executed interchangeably.

45. Arbitrary ENGLISH Page Numbers. It is often desirable to embed ENGLISH output within other (often manual) reports. Unfortunately, the page numbers on ENGLISH reports always start at one, which looks confusing when the report is preceded by other documentation. Provide a mechanism to allow control of the page numbers output in ENGLISH reports.

46. CHECK SPOOLER! Message. Inexperienced users are often confused by the CHECK SPOOLER! message and seem to have a hard time remembering what it means. Provide a mechanism to turn off the CHECK SPOOLER! message, or keep the text in the ERRMSG file.

47. ISTAT and HASH-TEST Graph Suppression. It is often desirable to see the statistics generated at the end of the ISTAT or HASH-TEST output, even though the histogram may go unexamined. Generating the histogram for files with small modulos is no great penalty, but having to print or page by the histogram for files with large modulos, just to get to the statistics at the end, is a nuisance. Provide an option allowing the histogram for ISTAT or HASH-TEST to be suppressed.

48. Backspace Speed During Input. Even when the system is heavily loaded, there is no noticeable limitation on the rate at which keys may be typed, except for the backspace key, which can be very slow when the system is busy. Improve the speed of the backspace key.

An Introduction to ENGLISH

Part 1: Jargon

A series of articles is introduced for the beginning user of Microdata's inquiry and report generation language called ENGLISH. Some jargon and concepts are described to help prepare the reader for actually using ENGLISH commands.

This is the first in a series of articles that will be devoted to explaining ENGLISH, Microdata's computer language for retrieving and listing computer files. The word "files" is an example of the kind of jargon and buzzwords that frequently appear whenever someone tries to learn about using a computer. This initial article in the series will be especially concerned about identifying and describing some of the jargon that must be understood in order for you to be comfortable with ENGLISH. All of these articles are intended for readers with no previous computer experience, but at the same time they are designed to allow you to quickly start using ENGLISH and to begin getting useful results out of the computer.

The articles assume you have access to a computer (to try out some of the examples) and that the computer is made by Microdata. If you are instead using a "Microdata compatible" system, such as an Ultimate computer, or an Evolution computer, or other equipment, then a computer language very similar to ENGLISH will still be available for you to try, but it will probably be named something different. Also, your computer may have a few differences from the Microdata brand, so it may not always operate exactly as described here, but those kinds of differences will not happen very often; they generally won't start to be noticeable until we get farther along in this series of articles.

To actually use ENGLISH, you must **log on a terminal** and **input a command** to **list a file**. The phrases in bold type are jargon.

Let's examine each piece of jargon and find out what it means.

☐ **Log on a terminal.** To log on a terminal is to announce to the computer that you are going to begin using one of its terminals. A terminal is the combination keyboard/TV screen that gives you access to the computer. A number of different terminals are probably wired to your computer, and they are all probably used by a number of different people at your company. Depending on how your terminal was left by the last person to use it, the complete procedure for sitting down and starting to use the terminal can sometimes involve a lot of steps, so it's probably best to have someone familiar with the procedure to help you the first time you try it. And because you might be using any one of a number of different possible brands of terminal, we will not go into too much detail about the actual steps (again, rely on help). But in general, the log on procedure is:

1. Turn on the terminal's power and let it warm up.
2. Hit the RETURN (or ENTER) key once or twice.
3. The display screen should show the message "LOGON PLEASE:".
4. Type in your account name and hit the RETURN key. The account name is how you will be identifying yourself to the computer. The people who manage your computer should be able to provide you with an account name to use. If you are managing the computer, use SYSPROG for the account name.
5. If your account is protected with a password for security reasons, the computer will display "PASSWORD:". Type in your password (it should have been given to you by whoever gave you your account name) and hit the RETURN key.
6. After a few seconds, the display should show a colon (:), indicating the computer is ready to accept your commands.

You may find when you sit down at a terminal that someone has already done the first step or two, or maybe even all the steps.

☐ **Input a command.** To input a command means for you to type in one or more words in response to the colon displayed by the computer on the screen. After you type in a few words, you can hit the RETURN key to indicate you've finished your command, and to cause the computer to obey the command. In the same way you can command a person by telling them to "GO BUY A CAR", you can command a computer to perform certain tasks by typing in words like "SUM THE PAYROLL FOR THIS MONTH". Whenever you type in a command and hit the RETURN key, the computer will either perform the command you requested, or it will display some kind of message in an attempt to explain to you why it couldn't perform the command. In either case, when the computer is finished with your command, it will display another colon, indicating it is ready to accept another command from you. Some commands are long and consist of a number of words. Other commands are just one word. Try typing in whatever comes to mind and see how the computer reacts. Try the one-word commands WHO, CHARGES and OFF and see if you can figure out what they cause the computer to do.

☐ **List a file.** There is a certain group of available computer commands (called ENGLISH commands) which can cause the computer to list a file. A file in a computer is just a collection of related information stored in the computer's memory. In the same way that there can be an employee file on paper (just a collection of sheets of paper all containing information about employees), or a paper file of payroll checks, there can be a computer employee file and a computer check file. It just happens that instead of storing the numbers and data on paper, the computer stores the information (the file) in its electronic circuits. The commands that list a file are simply asking the computer to display (on the screen) or print (on the computer's printer) the contents of the file it has stored in its memory. Some fancy versions of commands can cause the computer to sort the file, or total up certain numbers in the file, or to do all kinds of special processing for us. A simple example of a command that lists a file is the two words "LIST ACC", which means "list the ACC file". Go to a terminal, get the colon prompt, and type in the command LIST ACC to see what the computer does.

In the same way that a filing cabinet can hold a number of different files (say, purchase orders in one drawer, sales orders in another drawer), the computer can also store a number of different files in its memory. Let's carry the analogy between the filing cabinet and the computer a little further. If we examine one of our paper files (a cabinet drawer) in detail, we might find that the file is really a collection of separate pieces of paper. For example, a purchase order file would consist of a collection of paper purchase orders, with typically each purchase order recorded on a separate sheet of paper. In our computer, each such record (whether it's a purchase order or sick leave report or whatever) is also separately stored in a file, and each "sheet" is called an **item**.

If we look at each sheet in our paper files, we usually find a name or number on every sheet that is different from all other sheets and is used to keep the file in order. For example, each purchase order would probably be stamped with a unique purchase order number, while maybe each card in the stock file is sorted and identified by an inventory part number. In the computer, the special number or characters that are used to identify each item in a file and to tell it apart from all others is called the **item identifier**.

A sheet from a paper employee file is likely to be covered with all kinds of boxes filled in with data about the employee, such as a name, address, telephone number, social Security number, and so on. In the computer, each item in a file is also divided up into boxes for holding such data. These boxes are called **attributes**.

Let's review the jargon we've learned for the way the computer stores data in its memory. The computer can store a number of different **files** (purchase orders, sales orders, employee records...). Each file consists of a number of different **items** (first purchase order, second purchase order, third purchase order...). Each item has a unique **item identifier** (PO3167, PO3168, PO3169...) that tells it apart from all other items in the file. Each item consists of a number of different **attributes** (purchase order date, vendor name, purchase order cost...) that hold all the data for the item.

It turns out that, unlike a file drawer or a piece of paper, the computer has no special limits on how much data it can store. As long as enough computer memory is purchased, the computer can store any number of files, and each file can contain any number of items, and each item can contain any number of attributes.

In the next installment, we will learn more ENGLISH commands, and find out how to start extracting data from computer files to create reports.

Some New Subscribers

Michael L. Verkler
RW Data Center
Salt Lake City, UT

Leslie Keene
Sega/Gremlin
San Diego, CA

Bill Irvine
Parthenon Metal Works
La Vergne, TN

Jeff Harbour
Tropical Circuits
Ft. Lauderdale, FL

command files

Command files are typically the "glue" that holds the components of a software system together. Microdata's command file capability is called PROC, Prime offers Perform, and so on. In this regular department, Pragma will be presenting concepts and techniques to help installations squeeze the most out of their command file processors.

PROC Conversions

Proc examples are presented, demonstrating the use of input and output conversions to allow special computations and logic tests.

A powerful but seldom used capability of Microdata's PQN procs is the specification of input and output conversions in IBH, IH, L and T commands. For example, the following proc is a simple and convenient program for converting dates to and from internal form:

```
  DATE.CONV
001 PQN
002 T %2:D2-:
003 T %2;D;
```

The proc will obligingly convert the date supplied when the proc is invoked, regardless of whether a date in external form or internal form is typed, as shown by these two examples of its use:

```
:DATE.CONV 2/5          :DATE.CONV 4419
2/5                     02-05-80
4419
                        :
```

With DATE.CONV as a guide, it should be obvious how to create a similar proc, say TIME.CONV, for converting time values.

Being able to convert dates to a standard output format is also handy for creating clean, informative headings in ENGLISH statements, such as in this proc that accepts two input dates:

```
001 PQN                 011 IF # A RTN
002 RI                  012 H AND<=
003 HLIST INVOICES      013 A"
004 OSTART DATE+        014 H HEADING "INVOICES
005 IP?                 015 B
006 IF # A RTN          016 IH%1;D;
007 H WITH DATE>=       017 IH%1:D2-:
008 A"                  018 A\
009 OEND DATE+          019 H TO
010 IP?                 020 IH%2;D;
021 IH%2:D2-:           024 P
022 A\                  025 RTN
023 H"
```

Assume that a small data file exists called MONTHS with 12 items, each item identifier being a month number, and attribute one in each item being the corresponding month name (JANUARY, FEBRUARY, and so on). Then a similar technique can be used to include month names in report headings:

```
001 PQN                 009 H WITH MONTH
002 RI                  010 A"
003 OMONTH+             011 H HEADING "INVOICES FOR
004 IP?                 012 IH%1:TMONTHS;X;;1:
005 IF # A RTN          013 A\
006 IFN A<1 RTN         014 H"
007 IFN A>12 RTN        015 P
008 HLIST INVOICES      016 RTN
```

Since frame address 50BB provides the familiar WHO output consisting of a port number and account name, a proc can be written to insure it will only be executed at a certain port number:

```
001 PQN
002 RI
003 IBH%1:U50BB:
004 IBH%1:G 1:
005 IF A # 30 XYOU'RE NOT ON PORT 30!
006 C WE'RE ON PORT 30, CONTINUE...
```

With this simple variation:

```
001 PQN
002 RI
003 IBH%1:U50BB:
004 IBH%1:G1 1:
005 IF A # SYSPROG XYOU'RE NOT ON SYSPROG!
006 C WE ARE SYSPROG, CONTINUE...
```

we have a final example of a proc that will only work if invoked by a certain account (in this case, SYSPROG), which is particularly useful for security purposes if multiple accounts need or want to share the same Master Dictionary.

The few examples presented here are only the tip of the iceberg of possible ways in which proc input and output conversions can be put to work. The next time a proc is about to be abandoned or redesigned because "it can't quite do the job", always give one more look to see if perhaps the missing proc ingredient is just a simple conversion. P

VANILLA The No-Frills Manufacturing System

Part 2: Bill of Material Input

The second in a series of articles on the design and implementation of Vanilla, a software system for manufacturing, is presented. Software and techniques for the creation and maintenance of a bill of material file are described.

This is the second in a series of articles devoted to the design and implementation of Vanilla, a software system for a hypothetical manufacturing company. The first installment in the series (Pragma No. 1, page 15) presented a program for creating and changing entries in the part master file. In this installment, a program for updating bills of material will be the subject for discussion.

A bill of material lists the components of each assembly and subassembly in a product. It is the Engineering document which defines the product's structure. Maintaining a bill of material file for all parts is an important aspect of any manufacturing system that includes MRP, since bills of material guide the MRP explosion and netting process that determines all requirements against inventory.

A company's Engineering department will typically have a large number of requirements concerning the acceptable content and structure of the bill of material file. Among the items that must be considered when implementing a data processing system that supports a bill of material file are:

COMPONENTS PER BILL. What is the maximum number of different component parts that can appear in an assembly's bill of material? *In Vanilla, the software includes no code that limits the possible number of line items (components) in a bill. Limitations on record sizes are only those that are imposed by the particular implementation of the operating system being used. (In general, this will be true of all Vanilla application programs that happen to deal with data records that include multiple line items.)*

DUPLICATE PARTS. Can a component part number appear more than once in an assembly's bill of material, or must all occurrences of a part be consolidated into one line item? *In Vanilla, the same part number is allowed to occur any number of times in a bill.*

QUANTITY PER. Are all component quantities always numeric? Are quantities ever negative, zero, or fractional? *In Vanilla, the only requirement of quantities is that they be numeric.*

UNITS OF MEASURE. Can a part with one standard unit of measure be called out with some other unit of measure in a bill of material? *In Vanilla, a part is documented, stocked, kitted and built all with the same unit of measure (as originally described in Vanilla's part master specifications), so units of measure are always implied by the part number and are only found in the part master file. Similarly, only one revision is maintained by Vanilla for each part, so a bill's revision is also implied by the revision found in the part master file.*

COMPONENT ORDER. Are component part numbers kept in sorted order in the bill of material? Is the order automatically maintained when new components are added as a result of Engineering changes? *In Vanilla, the order of component parts in a bill of material is always as entered manually by the operator, with no sorting taking place. Note that since Engineering changes in Vanilla are maintained by manual updates to the part master and bill of material files, as opposed to automatic file changes caused by interpretation of some separate "Engineering change order" file, there is no inclusion in Vanilla's bill of material file of any data for the tracking of Engineering changes.*

ALTERNATES. Do bills of material include alternate part numbers for components? How many alternates can appear? Are alternates included as separate line items, or as additional

Continued on page 44

PRAGMA • ISSUE 2 • NOVEMBER, 1982 • 207 Granada Drive • Aptos, CA 95003

GET.BM PROGRAM LISTING

```
001 EQU BM$PART TO 1
002 EQU BM$DESC TO 2
003 EQU BM$QTY TO 3
004 EQU BM$ALT TO 4
005 EQU BM$NOTE TO 5
006 *
007 EQU PM$REV TO 1
008 EQU PM$DESC TO 2
009 EQU PM$UM TO 3
010 *
011 EQU item TO 4
012 EQU part TO 5
013 EQU sho.rev TO 6
014 EQU desc TO 7
015 EQU qty TO 8
016 EQU sho.um TO 9
017 EQU alt TO 10
018 EQU note TO 11
019 EQU command TO 12
020 EQU sho.part TO 13
021 EQU sho.desc TO 14
022 EQU sho.qty TO 15
023 EQU sho.alt TO 16
024 EQU sho.note TO 17
025 *
026 OPEN "BM" TO BM.FILE ELSE STOP "VAN1"
027 OPEN "DF" TO DF.FILE ELSE STOP "VAN2"
028 OPEN "PM" TO PM.FILE ELSE STOP "VAN3"
029 READ SCR FROM DF.FILE, "#GET.BM" ELSE STOP "VAN4"
030 *
031 AT.ERR = @(0,23):CHAR(27):"K"
032 *
033 10 PRINT CHAR(12):
034 20 INPUT BM.ID USING SCR, "" ELSE STOP
035 BM.ID = BM.ID<1>
036 READU BM FROM BM.FILE, BM.ID LOCKED
037   PRINT AT.ERR:"Bill group locked!":
038   GO TO 20
039   END ELSE BM = ""
040 PRINT AT.ERR:
041 ITEM = 1 ; NOTE.PTR = 1
042 COMP.REV = "" ; COMP.UM = ""
043 I = 1 ;* Load data for each component
044 LOOP
045  BM.PART = BM<BM$PART,I>
046  HAS.PART = (BM.PART # "")
047 WHILE HAS.PART OR (BM<BM$DESC,I> # "") DO
048  IF HAS.PART THEN
049    GOSUB 430 ;* Get PM item
050    COMP.REV<I> = PM<PM$REV>
051    BM<BM$DESC,I> = PM<PM$DESC>
052    COMP.UM<I> = PM<PM$UM>
053    END
054  I = I+1
055 REPEAT
056 GOSUB 360 ;* Show current item
057 GOSUB 320 ;* Show note
058 *
059 210 * Get part number
060 OLD.PART = BM.PART
061 INPUT BM.PART USING SCR, BM.PART AT part ELSE
062   IF ITEM > 1 THEN ITEM = ITEM-1 ; GOSUB 360 ;* Show previous item
063   GO TO 300
064   END
065 BM.PART = BM.PART<1>
066 BM<BM$PART,ITEM> = BM.PART
067 IF BM.PART # "" THEN
068   IF OLD.PART # BM.PART THEN
069     GOSUB 430 ;* Get new data
070     COMP.REV<ITEM> = PM<PM$REV>
071     BM<BM$DESC,ITEM> = PM<PM$DESC>
072     COMP.UM<ITEM> = PM<PM$UM>
073     GOSUB 360 ;* Show new data
074     END
075   GO TO 230 ;* Skip to quantity
076   END
077 220 * Get description. Can't enter here unless part is null
078 OLD.DESC = BM.DESC
079 INPUT BM.DESC USING SCR, BM.DESC AT desc ELSE GO TO 300
080 BM.DESC = BM.DESC<1>
081 BM<BM$DESC,ITEM> = BM.DESC
082 IF (BM.DESC="") AND ((OLD.DESC#"") OR (BM.PART="")) THEN
083   * Description deleted
084   DEL BM<BM$PART,ITEM>
085   DEL COMP.REV<ITEM>
086   DEL BM<BM$DESC,ITEM>
087   DEL BM<BM$QTY,ITEM>
088   DEL COMP.UM<ITEM>
089   DEL BM<BM$ALT,ITEM>
090   GOSUB 360 ;* Show new item
091   IF BM.DESC # "" THEN GO TO 210 ;* Try again
092   END
093 IF BM.DESC = "" THEN
094   IF ITEM > 1 THEN ITEM = ITEM-1 ; GOSUB 360 ;* Show previous item
095   GO TO 260 ;* Skip to note
096   END
097 230 * Get qty
098 LOOP
099   INPUT BM.QTY USING SCR, BM.QTY AT qty ELSE GO TO 300
100   BM.QTY = BM.QTY<1>
```

GET.BM LISTING CONTINUED

```
101 WHILE NOT(NUM(BM.QTY)) AND (BM.QTY#"") DO
102   PRINT AT.ERR:"Quantity must be numeric!":
103 REPEAT
104 PRINT AT.ERR:
105 BM<BM$QTY,ITEM> = BM.QTY
106 250 * Get alt
107 INPUT BM.ALT USING SCR, BM.ALT AT alt ELSE GO TO 300
108 BM.ALT = BM.ALT<1>
109 BM<BM$ALT,ITEM> = BM.ALT
110 ITEM = ITEM+1
111 GOSUB 360 ;* Show new item
112 GO TO 210 ;* Ask for next part
113 260 * Get note
114 LOOP
115  INPUT NOTE USING SCR, BM.NOTE AT note ELSE GO TO 300
116  NOTE = NOTE<1>
117 UNTIL NOTE = "" DO
118  BM<BM$NOTE,NOTE.PTR> = NOTE
119  NOTE.PTR = NOTE.PTR+1
120  BM.NOTE = BM<BM$NOTE,NOTE.PTR>
121 REPEAT
122 DEL BM<BM$NOTE,NOTE.PTR>
123 NOTE.PTR = 1
124 GOSUB 320 ;* Show note
125 *
126 300 * Get command
127 GOSUB 360 ;* Refresh
128 PRINT AT.ERR:
129 LOOP
130   INPUT COMMAND USING SCR, "" AT command ELSE RELEASE ; GO TO 10
131   COMMAND = COMMAND<1>
132 UNTIL COMMAND = "FI" DO
133   PRINT AT.ERR:
134   IF COMMAND # "" THEN COMMAND = ICONV(COMMAND,"MDO")
135   BEGIN CASE
136   CASE COMMAND = ""
137     IF (BM.PART="")AND(BM.DESC="") THEN ITEM=1 ELSE ITEM=ITEM+1
138     GOSUB 360 ;* Show new item
139     IF BM.NOTE="" THEN NOTE.PTR = 1 ELSE NOTE.PTR = NOTE.PTR+1
140     GOSUB 320 ;* Show note
141   CASE COMMAND = 1
142     GO TO 210
143   CASE COMMAND = 2
144     IF BM.PART # "" THEN
145       PRINT AT.ERR:"Can't change part master descriptions!":
146       END ELSE GO TO 220
147   CASE (COMMAND = 3) OR (COMMAND = 4)
148     IF (BM.PART#"") OR (BM.DESC#"") THEN
149       BEGIN CASE
150       CASE COMMAND = 3 ; GOTO 230
151       CASE COMMAND = 4 ; GOTO 250
152       END CASE
153       END ELSE PRINT AT.ERR:"Enter part or description first!":
154   CASE COMMAND = 5 ; GO TO 260
155   CASE 1
156     PRINT AT.ERR:"What?":
157   END CASE
158 REPEAT
159 *
160 I = 1 ;* Don't save descriptions for items with part numbers
161 LOOP
162   HAS.PART = (BM<BM$PART,I> # "")
163 WHILE HAS.PART OR (BM<BM$DESC,I> # "") DO
164   IF HAS.PART THEN BM<BM$DESC,I> = ""
165   I = I+1
166 REPEAT
167 WRITE BM ON BM.FILE, BM.ID
168 GO TO 10
169 *
170 320 * Show note
171 BM.NOTE = BM<BM$NOTE,NOTE.PTR>
172 INPUT IGNORE USING SCR, BM.NOTE AT sho.note ELSE NULL
173 RETURN
174 *
175 360 * Show current item
176 INPUT IGNORE USING SCR, ITEM AT item ELSE NULL
177 BM.PART = BM<BM$PART,ITEM>
178 BM.REV = COMP.REV<ITEM>
179 BM.DESC = BM<BM$DESC,ITEM>
180 BM.QTY = BM<BM$QTY,ITEM>
181 BM.UM = COMP.UM<ITEM>
182 BM.ALT = BM<BM$ALT,ITEM>
183 INPUT IGNORE USING SCR, BM.PART AT sho.part ELSE NULL
184 INPUT IGNORE USING SCR, BM.REV AT sho.rev ELSE NULL
185 INPUT IGNORE USING SCR, BM.DESC AT sho.desc ELSE NULL
186 INPUT IGNORE USING SCR, BM.QTY AT sho.qty ELSE NULL
187 INPUT IGNORE USING SCR, BM.UM AT sho.um ELSE NULL
188 INPUT IGNORE USING SCR, BM.ALT AT sho.alt ELSE NULL
189 RETURN
190 *
191 430 * Get PM data
192 READ PM FROM PM.FILE, BM.PART ELSE PM = ""
193 RETURN
194 *
195 END
```

Ⓟ

Continued from page 19

that we could or should at this point in time. We also prepare our internal financial statements on the computer. We prepare our budgets on the computer. A lot of people ask us why don't we do the preparation of tax returns on our own computer, but we found that the area is always constantly changing. We had two or three tax law changes over the past two years, and it's just too hard to keep up. So we subcontract out those services versus doing them ourselves. It's like a make or buy decision.

Pragma: What are some of your newest software projects?

Kempter: One priority right now is to implement the word processing software that we just purchased from Microdata, which is called Wordmate. We want to get up and running and prepare various types of repetitive letters that we use in the accounting profession. These are letters such as engagement letters that spell out the services that you are going to provide for the client. One project we have going also is trying to see if out there is a good manufacturing package that we can work with. Most are so sophisticated that they really wouldn't suit the needs of our clients. They are just too sophisticated, and when you're dealing with businesses the size that we are, they just can't relate to it. They just want the bread and butter. Like right now, we're designing a parts inventory control system. We want to control our receipts, the issues, and in integration with that, the job cost system. They're not concerned with the economic order quantity and all these various esoteric pieces of data that are available. They have a hard enough time doing their accounting. You really have to keep it simple. I can't find anything out there that's simple.

Pragma: What is the state of the Wordmate project right now?

Kempter: Right now we're waiting for Microdata to release that to us so it can be implemented on the Sequel. I understand that it's up and running on the Microdata 8000 series and it's very shortly going to be released to be run on the Sequel that we have.

Pragma: Have you actually already purchased it and are definitely going to be using it?

Kempter: Yes, we have purchased the software along with all the special hardware that's required. There's the special dual function terminal that can do both word processing and data processing and then there's a special letter quality type of printer that's required, plus various auxiliary devices that go along with the printer.

Pragma: Did you do any kind of evaluation of Wordmate versus other alternatives?

Kempter: Yes, we looked at Wordmate versus Lanier and Lexitron, and we thought since we can integrate word processing and the data processing all in one computer, that would be a better route to follow.

Pragma: How involved was that evaluation?

Kempter: Well, it wasn't really that detailed an evaluation. We looked at the type of work that we wanted done on our system. As I said before, we had various repetitive letters and memorandum that we generate, so we saw what was being done in the secretarial pool. We saw the main use was the preparation of financial statements. Then after looking at that, we had each vendor come in and give us a demonstration. We thought, looking at the entire thing, that Wordmate, because of its integration with the data processing part, would really help us. For example, we have our customer master file stored in the computer. So when we want to prepare our bills, we just extract all that information from the customer master file and then use the word processor to actually prepare the bills. We have mailings to our clients which are really quite extensive. Each month we send them our client newsletter. When there's a change in the tax laws, we have a mass mailing, and things of that nature. We thought we could integrate a lot of this information, take it right from our existing data base.

Pragma: Besides seeing the vendor demonstrations, did any Galinski Hamburg personnel actually sit down and try using and comparing the various systems?

Kempter: The secretaries went to school for a period of about two months, and they worked on the Lanier and Lexitron at one of the local community colleges. Then they went to Microdata to work on their units that they had on the 8000 series, and they seemed to like the Microdata system.

Pragma: Did you talk to any other sites that are using Wordmate?

Kempter: No, because we signed a contract just about the time that Wordmate was announced. Our people have talked to the customer engineers and they seem to feel that the sites that have it really like it so far.

Pragma: What do you do to keep your projects in control and on time?

Kempter: I think the key thing is planning. Proper planning prevents poor performance. Another key is to try to get the programmers involved as soon as possible in the project, so that if you're out there meeting with the clients, the programmers are there to see the personalities involved and what the client is looking for. As far as monitoring what they're doing, I force them to think ahead. Every week they have to submit to me what they'll be doing for the next four weeks. They're constantly not only thinking about today, but a week from now, two weeks from now, and so on. When they get done with a certain aspect of a program, we have them go through a dry run with myself and our data processing manager.

Pragma: How do you insure that what is developed will work?

Kempter: By having the client submit as much test data as possible. This is a real key in software development. The only way we really feel you get test data is to get real live data. Right at the beginning of the project, we try to see what the client is doing on a manual basis, get that documented as much as possible, and then, once the system is designed and programmed and before it's operational, we make the client submit to us an awful lot of actual live data that we work with.

Pragma: How do you feel about Galinski Hamburg end user documentation and programmer documentation?

Kempter: Programmer documentation is good, because I think that the Microdata system is almost self-documented, and we have some strict standards established for our data processing documentation. Our user documentation leaves a lot to be desired. At times it just isn't there, or is just too technical. We have a long way to go in that particular area, a long way.

Pragma: How do you handle reviews of your data processing personnel?

Kempter: We do that on an annual basis, and I don't know if it's unique or not, but I think it's a little different. We'll have the manager review the employee, but also the employee review his own performance. And they do it separately. So for example, when I review the data processing manager, I'll do it independently and he'll do it independently, then we'll get together and compare notes. It's an amazing process, and it insures that you are communicating with each other, that you're

Continued on page 42

queries

If you have questions or if you have answers, send either to Pragma for publication — both are eligible for Pragma's author payment awards.

The first four Queries appeared in Pragma #1.

5. The system documentation does not contain any guidelines indicating what an account's workspace parameter in attribute eight of SYSTEM account definition items should be. What are the pros and cons of large and small workspaces? How does one determine the optimum workspace size for a given account?

6. With the 4.1 spooler, the SP-ASSIGN HS command will successfully set up a hold file. But when later using action code 3 in the SP-JOBS display to cancel the HS option in an attempt to make the job print, nothing happens. How can a job that has had the HS option cancelled be made to print?

Use SP-JOBS action code 8, or the SP-EDIT verb, and then give the P command to print the job.

7. For reasons our customer engineer cannot explain, he has been taught to use a verb he calls CHOP before running the various ATP diagnostics. He manually creates the verb with the Editor, placing a P in attribute one, and 2118 in attribute two. Also, he has been taught to enter six zeroes with the debugger at 280.135;3 when running the BLAST diagnostic under the 4.1 release. What does CHOP do, and why is the BLAST patch necessary?

8. What does ERRMSG item [13] DATA LEVEL DESCRIPTOR MISSING mean?

This message results when trying to list a file that does not have an item named DL/ID in the file's dictionary. The DL/ID item is intended to point to the frames on disk that make up the data level portion of any file (as opposed to the dictionary level portion). Some files, such as SYSPROG's TSYM file, are typically only used as a dictionary level file, and so software that accesses the file may never bother to maintain an item named DL/ID. But most files do have a DL/ID item in their dictionary, which should remain there indefinitely, unless clobbered by some sort of system or programming bug, or deleted through careless use of the Editor. If the error message is suddenly encountered while trying to list some file, use the DICT prefix in the command to try and avoid the error. For example, use LIST DICT TSYM instead of LIST TSYM.

9. Other than using the CPU console switches, is there any way to successfully log onto (break into) a Microdata 8000 system that has unknown passwords set for every account?

10. How does one determine the current percentage of total disk space still available, when running under 4.1?

To determine the percentage of disk space still available for files at any given moment, first give the WHAT command in the SYSPROG account to determine the total file space in the configuration. Using the frame numbers displayed by the WHAT command, subtract the FRAME ID OF THE BASE OF THE SYSTEM DICTIONARY from the MAXIMUM FRAME ID FOR THIS SYSTEM and add one, to get the total frames available in the configuration. Then give the POVF command to find the number of frames currently unused. The TOTAL NUMBER OF CONTIGUOUS FRAMES displayed by POVF plus the number of LINKED frames, if any (displayed at the top of the first column output by POVF), is the number of frames currently unused. This number, divided by the configuration total determined from the WHAT command, is the percentage of disk that is still unused. (Actually, a small number of frames included in the POVF display may be grabbed for account workspaces and therefore not be available for files, but the total number of such frames is usually not significant.) Note that just because a frame is physically on disk in the file space area does not mean the frame is guaranteed to be used in a file or be free as available space, since if a D-pointer is deleted (say with the Editor), then those frames addressed by the D-pointer are lost to the system and not reusable until the next file restore. Only the DELETE-FILE verb can properly delete a file and return its frames to the POVF list. If the above calculations indicate only a small number of unused frames are left on the disk, but the STAT-FILE report totals do not show a balancing large total number of frames used, then the system is probably overdue for a file restore, since it appears some D-pointers have been deleted, resulting in the loss of frames. In any case, it's always a good idea to periodically monitor disk consumption, so that extra mass storage can be acquired in plenty of time.

11. On some of our ADDS terminals, why does the DATA/BASIC statement INPUT X,10__ cause asterisks to display when the operator attempts to type more than ten characters, instead of stopping the cursor and beeping like our Prisms and other ADDS units?

12. When the WHERE (S) command is given, a column showing LINKS is output, but the documentation does not explain what LINKS are. What are LINKS?

13. How can a program written in DATA/BASIC be made to lock individual file items for updating, instead of locking whole groups?

The first thing to do is make sure that item locking is really necessary, since going to all the trouble of setting up item locks may not really be worth it, or even required. Two users

Continued on page 34

the computer room

A well-run computer room is the sign of an efficient data processing organization, and an important requirement in any computer room is good documentation — documentation regarding the care and use of peripherals, system error recovery, maintenance procedures, and a myriad of other details regarding day-to-day computer operations. In this regular department, Pragma will be presenting documentation, ideas, and techniques for guaranteeing a smooth-running computer room.

The Trouble Tree

A document designed to allow inexperienced personnel to diagnose and correct system malfunctions is described.

No computer should be without a Trouble Tree, which is a document for diagnosing hardware and software malfunctions (see the sample listing on the opposite page). The document is organized as a series of questions that can be answered yes or no. Either answer leads to a more refined question, in an attempt to narrow down the cause of the problem so a solution can be recommended. Since each question "branches" off to two more questions, the complete document forms one large binary tree (a structure familiar to most programmers), hence the name Trouble Tree.

Trouble Trees are easy to maintain on a computer, although a hard copy should always be available in case the computer goes down. By giving each question in the Tree a number, a file of Trouble Tree questions can easily be established by letting question numbers be item identifiers, and by storing the question text as an attribute. Two other attributes can be used to store the numbers of the two questions next in line if the answer is yes or no. The Editor can be used to maintain the file of questions, and ENGLISH is useful for printing out the Tree in sorted question order. Since Trouble Trees are frequently restructured as questions are added or changed because of experience gained while diagnosing new problems, having the questions available as items in a file, instead of as one large text document, often makes for easier editing of the Tree.

Trouble Trees are especially valuable because they allow inexperienced personnel to quickly diagnose and correct system malfunctions. A Trouble Tree should be comprehensive, covering all types of possible system failures. A complete Tree must address potential problems with any peripheral (such as terminals, tapes, or printers), problems with daily procedures (such as batch runs, FILE-SAVEs, and restores), problems with file structures (such as group format errors), and of course, complete catastrophes (such as a system crash). The Tree should be continuously checked and maintained: if a new problem is encountered that isn't already covered by the Tree, the Tree should be modified to include the problem so that if it should happen again, the solution can be quickly found. With a complete Trouble Tree at its disposal, an installation will discover that a number of problems formerly requiring the attention of special personnel can now be handled as a routine clerical matter.

TROUBLE TREE
SAMPLES

These are questions extracted from a Trouble Tree, demonstrating the sequence in which a terminal malfunction might be diagnosed.

STEP	QUESTION	IF YES GO TO	IF NO GO TO
0	The system has some kind of problem. Is the problem with one or more pieces of the hardware?	1	63
1	Something appears wrong with the hardware. Are one or more terminals failing to respond?	5	7
5	One or more terminals are failing to respond. Is the HALT light on the front of the system's console lit?	19	9
9	One or more terminals are failing to respond and the HALT light is not on. Is it only one terminal that is failing to respond while the other terminals appear all right?	13	25
13	Only one terminal is failing to respond. Be sure the terminal does not have the printer or tape drive attached and is simply waiting for the output device to be ready. The terminal is probably the problem. Make sure all switches are at proper settings and all cables are securely connected. If necessary, power the terminal off and back on to reset it. If the terminal still does not work, swap it with a terminal known to be working to determine if the problem is on the terminal side or on the computer/cable side. Does even a good replacement terminal known to have been working also fail to respond when attached to the port?	27	45
25	More than one terminal is failing to respond. The system has not halted but apparently is hung. Try the recovery from lost interrupts procedure posted on the wall of the computer room. Are all terminals now functioning normally?	36	39

Continued from page 31

sharing a file will attempt to lock the same group surprisingly seldom, in all but the smallest of files, so that item locking gives no particular advantage over the system's built-in group locking. Even if the file being shared is very small, picking a large modulo for the file usually spreads out the items enough so that there is only one item per group, and locking groups for updates then becomes the same as locking items. But if for some reason item locking is still desired, the effect can most easily be accomplished by setting aside a "lock byte" (typically attribute one) in each item of the file where item locking will be supported. Then any time it is necessary to update an item, the program should do a READU to momentarily lock the item (and its group) and check the lock byte. If the lock byte in the item is "set" (say, not null, by convention), then someone else has the item locked and the program must release the item (and its group) and try again later. If the lock byte is clear (null), then no one has the item locked, so the program should set the lock to some non-null character and write the item back to the file. In either case, the group containing the item will only be momentarily locked, and other processes can then immediately begin updating other items in the same group. When the program is finished with the item, it must remember to clear the lock byte. For example,

```
READU ITEM FROM ID ELSE ITEM = ""
IF ITEM <LOCK> # "" THEN
 ! ABORT, ITEM ALREADY LOCKED
 RELEASE ID
 GO TO 999
 END
! ITEM IS AVAILABLE, LOCK IT AND RELEASE GROUP
ITEM <LOCK> = "X"
WRITE ITEM ON ID
! NOW WE CAN TAKE OUR TIME UPDATING THE ITEM
ITEM <BALANCE> =ITEM <OWED> —ITEM <PAID>
! NOW CLEAR LOCK AND WRITE FINAL ITEM
ITEM <LOCK> = ""
WRITE ITEM ON ID
```

Technically, only a WRITEV would be necessary to set a lock, but there have been reports of a bug preventing WRITEV from unlocking groups. To avoid wasting a byte in every item in the file, a separate file can be set up for the locks, where the presence of an item identifier indicates the item is locked:

```
LOCKED = 1
READU IGNORE FROM LOCK.FILE, ID ELSE LOCKED = 0
IF LOCKED THEN RELEASE LOCK.FILE, ID ; GO TO 999
WRITE "" ON LOCK.FILE, ID
READ ITEM FROM DATA.FILE, ID ELSE ITEM = ""
ITEM <BALANCE> = ITEM <OWED> —ITEM <PAID>
WRITE ITEM ON DATA.FILE, ID
DELETE LOCK.FILE, ID
```

Having a separate lock file also avoids checking the whole data file just to see everything that's locked, if that kind of monitoring is required. If one lock file is to support multiple data files, then the item identifier in the lock file should consist of the data file name concatenated with the data item name. Generalized item locking routines can also add extras like recording which port and account has the item locked and so on, but those features might cause unacceptable delays and overhead.

14. The system's ACC file allows the disk reads and CPU time consumed at each session to be recorded, but how should those figures be translated into tangible costs? What is a good algorithm for computing dollar charges for a session based on the statistics gathered in the ACC file?

15. Releases 3.2 or 3.2B are familiar, but what are 3.2-1 and 3.2.3 and 4.1-23?

Many installations like to record the current number of patches they have installed in the release currently on their machine. This is typically done by placing the number of patches installed in ERRMSG item 335 after the release number found there, separated by a period or hyphen, resulting in release numbers like 3.2.3 (3.2 with 3 patches) or 4.1-23 (4.1 with 23 patches). Sometimes the prefix indicates the number of patch reports, not the number of patches.

16. How can a BREAK-ON column be suppressed from an ENGLISH output report?

Use a column width of zero in the V/MAX field (attribute ten) of the dictionary item for the word following "BREAK-ON".

17. What's the difference between Microdata's SYSPROG account and SP account?

The only possible differences between these two accounts are the parameters defined in the SYSTEM file's definition item for each account. In other words: retrieval and update lock codes, password, system privilege level, workspace size, and ACC file update flag. On a standard Sysgen tape as delivered by Microdata, the SP item is a Q-pointer to the SYSPROG account, and all of the parameters are identical for the two accounts except for the workspace size. So once a user logs on to either account, the same Master Dictionary ends up being used, and all system files and verbs are accessible from either account in an identical manner. Since SP and SYSPROG are two different account names, two different logon procs may end up being executed, if procs named SP and SYSPROG are stored in the Master Dictionary. Since SP has a smaller workspace, it will abort when trying to execute certain functions, such as doing a FILE-SAVE with an 8192 tape record size. Apparently the idea behind SP was to have an account that would allow access to the SYSPROG account, but which could be logged on faster because of the short account name and minimum workspace size.

18. What are the steps to create in one account a file that is a duplicate of some other file in another account?

Assume the two accounts are called ALPHA and BETA, with BETA being the account in which the new file is to be created. BETA must have at least SYS1 privileges to allow creating the file. From the BETA account, use the CREATE-FILE command to create the new file. (See the system's documentation for an exact description of the format of the CREATE-FILE command.) It's probably a good idea to use the same modulo and separation used by the old file, which can be looked up in the STAT-FILE report from the last FILE-SAVE. Then use the command SET-FILE ALPHA OLD (assuming the old file is named OLD), to set up a Q-pointer to the other account. Then use the command COPY DICT QFILE * to copy over all dictionary items. If the new file is named NEW, enter "(DICT NEW)" in response to the prompt "TO:" that is generated by the COPY command. Then use the command COPY QFILE * to copy all data items. Answer with "(NEW)" when prompted again with "TO:". This procedure should work fine as long as the OLD file isn't protected by retrieval codes. If the system won't allow the COPY because the file is access protected, the personnel responsible for system security will have to help.

letters

If you use a Pick system, Pragma wants to hear from you. Have you developed applications of interest to other users? Do you plan to acquire new hardware? What features would you like to see in Pragma? Are you active in any type of user group organization? Write Pragma today. All letters to the Editors are welcome, and as many as possible will be published in the Letters Department in every issue.

Computing Modulos

I would like to point out a deficiency with your technique for "Computing Modulos with ENGLISH" (Pragma # 1, page 29). The algorithm suggested does not work well for files containing items of an average size greater than 250 bytes. This is a common size for items on many Pick based systems. Consider a file of 1,000 items with an average size of 300 bytes. Your NEW.MOD correlative would calculate a suggested modulo of about 600. (For simplicity, the issues of divisibility by two or five and the quality of hashing distribution are not considered here.) If this suggested modulo of 600 is used, 40% of the data will go into overflow frames. The correct choice of modulo is 1,000. With either choice, the same amount of disc space will be consumed: 600 frames of primary space and 400 frames of overflow space in the first case, 1,000 frames of primary space in the second. The NEW.MOD correlative suggests a file size which requires many more disc accesses but does not consume less disc space.

Joseph H. Zeligs
Generation Research
San Diego, CA

We're not sure what other algorithm you prefer instead, but our experience has shown that the one demonstrated in the article results in good file performance. Actually, the article was never intended to promote any algorithm as being preferable over another. Instead, the article was designed to show that ENGLISH is quite capable of performing complicated computations, such as selecting new modulos. (The article also clearly showed how obscure the logic of long F-type computations can quickly get. It would be interesting to see the same algorithm expressed in A-type IF-THEN-ELSE form, for comparison.) The "Modulo Setting Program" article on page 6 of this issue is in a similar situation: even though the article deals with setting modulos, the intent is to show how to use an algorithm to set reallocation parameters. It doesn't really care what the particular algorithm is.

When it comes to actually suggesting how to compute "correct" modulos, we feel there are a number of pitfalls which make it difficult to find a perfect algorithm. Your suggestion implies that the primary concern in correctly sizing files is to reduce the number of items in overflow. Often, disk space is more of a concern than access time. 40% of each frame is wasted when filing 1,000 300-byte items with a modulo of 1,000. If the file grows to 10,000 items and we stick to a modulo of 10,000 to avoid overflow, two megabytes are being used for nothing. Reduce the modulo to only 3,333 and suddenly far less than a half-megabyte is wasted. One begins to discover that choosing modulos really involves trading time for space. Time can be wasted because of excessive frame faults to get to an item; space can be wasted because items are not packed well enough within multiples of 500 bytes. One also discovers that systems and files are often incredibly synergistic: the number of items is always changing; the size of each item varies; sometimes items are only accessed randomly; other times a SELECT is used and every frame is paraded through memory. Hypothesizing with nice round numbers, or ignoring the effect of one or two factors, can result in the selection of an algorithm for a situation that simply does not apply. What makes a modulo "correct" is its ability to meet some kind of specified time or space criteria, and a modulo algorithm should be considered only a contender until proven the best through the use of actual benchmarks. In the mean time, we like our original algorithm because it is quite simple, yet overall it has given good results regardless of the current size or contents of a file.
—Editors

16-Ways from Irvine

The following is in response to your Wish List item #18, "Changing Microdata Baud Rates" (Pragma #1, page 28). Our company now has in production a 16-way board that does not use DIP switches to set baud rates. The board is firmware controlled and has the following capabilities:

1. Baud rates from 110 to 19,200.
2. Baud rates can be set from a terminal or the system.
3. Automatic baud rate detection.
4. Complete hand shaking with peripherals.
5. Baud rates are stored in a non-volatile memory when power fails and are retrieved on power-up.
6. Users can write their own firmware for special communication requirements.

Dennis Keats
Irvine Computer Corp.
Costa Mesa, CA

A Switchbox for 32 Ports

A custom piece of hardware for the computer room, designed to allow the disconnection of terminals from a host computer, is described.

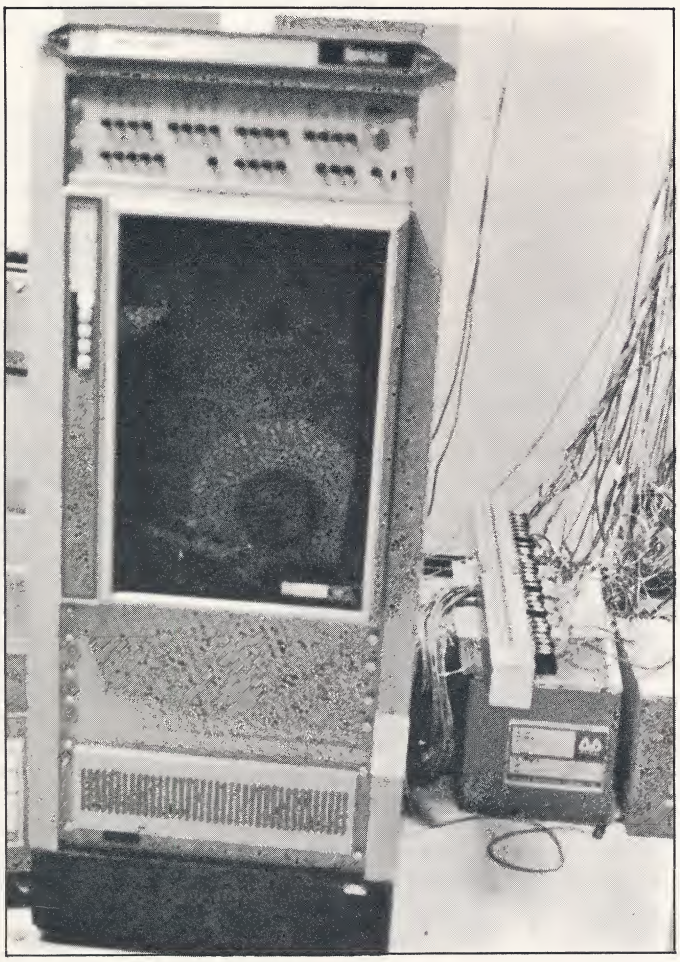
Most data processing installations try hard to keep the computer continuously available for its users, with a minimum of downtime. But there are occasions when it is desirable to *prevent* users from being able to access a system with their terminals, such as while recovering from a system crash (in order to give data processing personnel a chance to diagnose and verify the system is fully operational, so that the crash recovery will be smooth and uncomplicated). Or, for example, it is convenient to keep users off a system immediately after a file restore, to give the system operator a chance to do a SYSTEM-SETUP and other such chores. Or, if an installation is experiencing hardware problems, then a mechanism for quickly stopping user activity is a handy tool in case of an imminent failure (such as might be indicated by the sudden appearance of ampersands, a warning of disc errors).

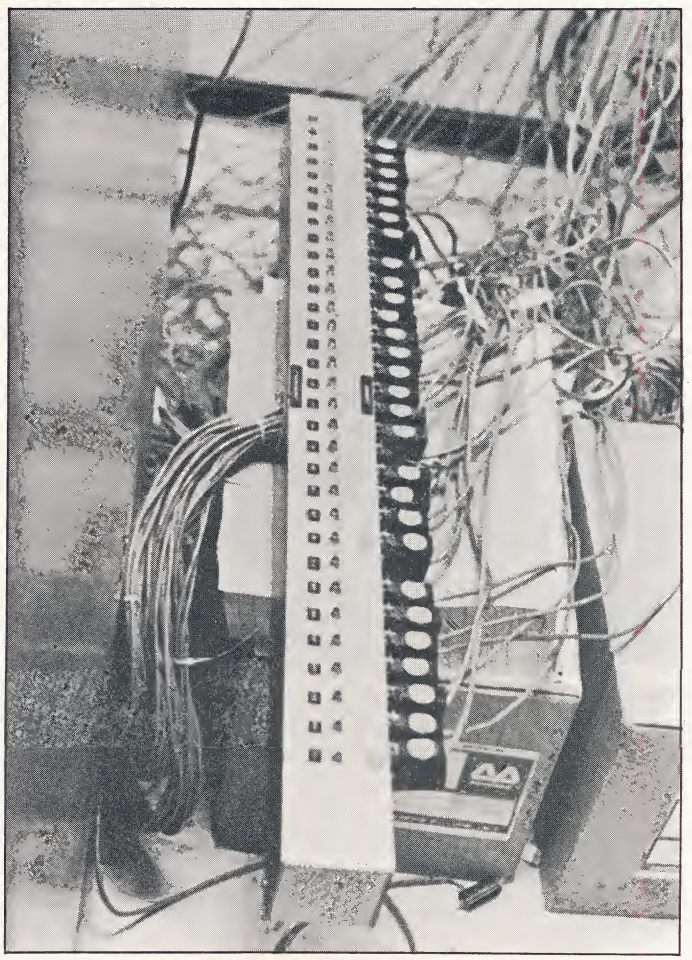
For users of Prime hardware, Microdata's Sequel, or other systems with software controlled baud rates, simply using a command to change a port's baud rate to some setting different than the terminal's rate is an effective way to cut off a user. And now that 4.1 establishes the date and time before the system is fully booted, some installations don't even need to worry about users getting online too quickly after a file restore. But many installations only have hardware controlled baud rates, or are not yet on the 4.1 release. Or, even if they are on 4.1, some sites still need to perform housekeeping tasks such as creating spooler queues, after a file restore but before users log on. Such sites still need a convenient, foolproof method of keeping users off a system.

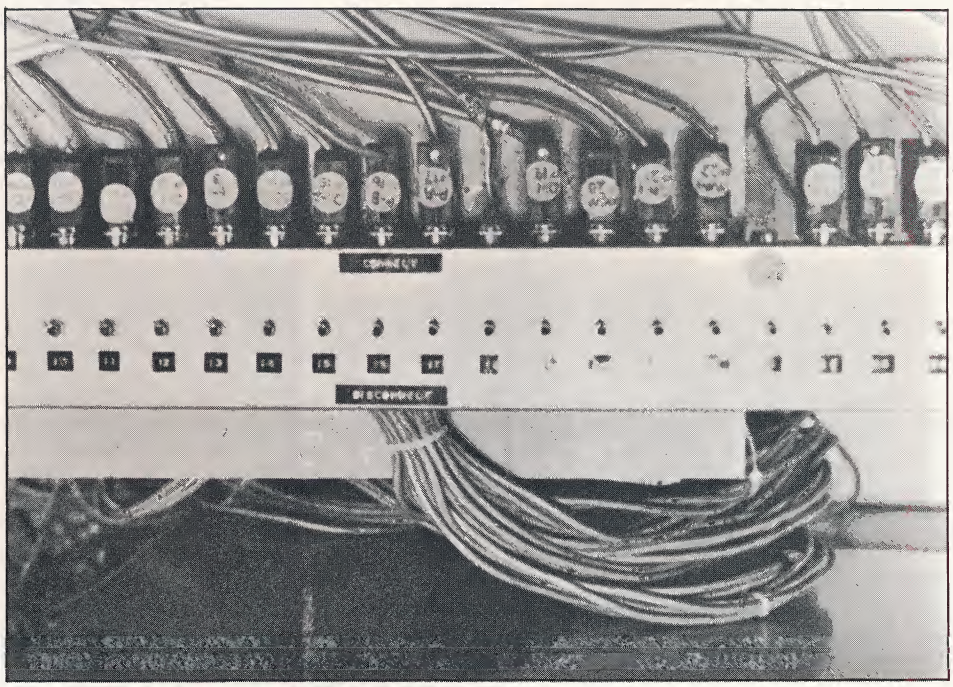
One Microdata installation's solution is shown in the photographs on the right: a custom-built switchbox for controlling up to 32 ports. The sheet metal switchbox measures three inches square, is a yard long, and is quite simple in design. Instead of plugging into the backplane of the computer, the cable from each terminal plugs into one of the 32 female RS232 connectors along one side of the switchbox. Each female connector continues as a relatively short cable that comes out the opposite side of the switchbox through a two inch hole, as part of a bundle of 32 cables that are plugged into the computer's backplane.

The switchbox cables match, and are a continuation of, the cables from each terminal. In other words, just like a terminal's 3-wire cable for sending, receiving, and ground, the switchbox cables each have a matching set of three wires. The difference is that the terminal's send line (pin 2 on an RS232 connector) is wired through one of the 32 toggle switches (Cutler-Hammer No. JMT-123B) along the top of the switchbox. By toggling the switch, the terminal can be effectively connected or disconnected from the system. Since the terminal's receive line (pin 3 on an RS232 connector, leaving pin 7 as the ground) bypasses the toggle switch as it runs through the switchbox, the terminal can continue receiving text and messages from the computer even if the terminal's sending capability has been switched off. (This is the main difference between disconnecting terminals by using such a switchbox versus simply unplugging cables directly at the backplane.) Also note that switching off a terminal only prevents further operator input. It does not halt a running process, such as a SORT that may have already started.

This particular switchbox has been in use for a number of years, and has given reliable service while saving a tremendous amount of time and trouble for data processing personnel on many occasions. Other installations may want to consider building a similar device.







Three views of the installed switchbox (clockwise from upper left): the switchbox alongside a Microdata 6000; a closer view showing the bundle of cables coming from the computer into the left side of the switchbox, and 32 connectors on the right side of the switchbox with cables leading to terminals throughout the building; a closeup of the switchbox.

Security and the DATA/BASIC Programmer

Methods are discussed for preventing the circumvention of system security measures by DATA/BASIC programmers at Microdata installations.

Most data processing managers at Microdata installations seem resigned to the idea that their programmers can always manage to bypass system security, and so the programmers are given (however grudgingly) free access to all files and data on the system.

Is that really the only choice? Or can accounts and files be configured in such a way that even a programmer cannot reach certain parts of the system? Would it be possible, for example, to set up an account for teaching introductory programming to complete novices, while insuring the system is completely safe from any intentional or accidental error a student might cause?

Let's list some of the weak points in the security of a Microdata system, and examine ways to prevent programmers from exploiting those weaknesses.

Access to the system debugger. With access to the debugger, a programmer can get to any frame on the system. Keeping an account at SYSO privileges solves this problem nicely, while also preventing tape access and Master Dictionary editing (two other potential security problems). Unfortunately, use of the DATA/BASIC debugger is also severely hampered when limited to SYSO privileges, so that program testing may be much more difficult. But surprisingly few programmers even bother learning how to use that debugger, and debugging can always be done through more traditional methods anyway (such as by the insertion of PRINT statements), so SYSO may not really be too high a penalty.

Access to the TCL. Although the Master Dictionary may need to contain a number of powerful verbs, such as EDIT, to allow practical program development to take place, being able to execute any form of such a command at the TCL level can easily become a security problem, since the programmer can use the Editor to edit any DL/ID pointer in any unprotected file dictionary in order to point to any frame on disk. Not every file in the programmer's account can be write-protected (after all, at least one file is going to be used to contain code newly written by the programmer), so the account must be configured with a logon proc that "traps" the programmer and prevents any access to the TCL. Using the BREAK key and typing "END" can be arranged to have no effect by including an "R" in attribute nine of the programmer's account definition item in the SYSTEM file, so that the logon proc is simply re-entered. This proc, typically a kind of "menu" of commands, must be sophisticated enough to be able to invoke limited forms of all the verbs the programmer needs to create and test programs. For example, to edit the BP file (but not the BP dictionary), to compile a BP item, to execute the item, to log off, and so on.

Access to certain DATA/BASIC statements. Once a programmer with only SYSO privileges has logged on and become trapped in a logon proc that allows only limited forms of program development commands, the only remaining way in which the programmer might circumvent system security is by using certain powerful DATA/BASIC statements. For example, OPEN and WRITE would allow a programmer to access and change the DL/ID item in the BP file of the programmer's own account; ICONV and OCONV allow transfers to any frame address; CHAIN "EDIT DICT BP DL/ID" is another simple way to get to a D-pointer quickly. The only way to prevent these kinds of tricks is to include in the proc controlling all of the programmer's actions an extra step that scans any program before it is compiled, to check for occurrences of CHAINs, or conversions at user mode addresses (which might appear as a variable name, not just a literal like "U8A32"), or OPENs of dictionary files, or any other perceived security violation.

What should be apparent by now is that a system can be made secure from programmers only by invoking extreme measures, which can drastically hamper the productivity of the programmer. A considerable number of improvements in this type of software environment are necessary before the programmer can be as productive as possible while not being a security threat.

games

For many people, computers are synonymous with games, now that the video game industry has become such a giant. Programmers frequently cut their teeth on small game programs, since such programs are often straight-forward and self-contained without a lot of complicated interfaces to files or other software. And, more than anything, games are simply entertaining and fun.

The Games Department will be making periodic appearances in Pragma. If you have a game program you would like to share, send it in for publication. If there's anything the Pragma staff has time to do, it's "performing rigorous software quality assurance" (in other words, trying out a new game on the computer).

COOKIE

It is 7:06 AM and the Production Control Manager has just arrived. She immediately goes to her desk, logs on her terminal, and gives the COOKIE command. After a few milliseconds, the Honeywell Ultimate at the other end of the building shoots back with IT IS BETTER TO WEAR OUT THAN TO RUST OUT. About a half hour later, the receiving clerk goes online and immediately types in COOKIE. The cursor spurts out a line and the display glows with BEWARE! ICICLES ARE EAVESDROPPERS. By 8:16, accounts payable personnel "eat" their COOKIE and get WHEN YOUR SPOUSE GOES OUT THE DOOR, LOVES COMES INNUENDO. Customer service calls order entry to tell them their COOKIE said DON'T LEAVE. YOU'LL BE HARD TO REPLACE AT THE SALARY WE'RE PAYING YOU. By 9:57, every user has used up their one COOKIE quota for the day.

COOKIE is a simple little fortune telling program (see the listing on page 40) that was inspired by the FORTUNE command available on most Unix operating system implementations. On Unix, the FORTUNE command just randomly selects and displays a one-line fortune from a rather small file of available fortunes. Since the FORTUNE command can be used as often as desired, most users repeatedly execute the command until they grow bored or until they have managed to display every fortune. Then the novelty wears off and they lose interest.

On the other hand, COOKIE is deliberately designed to prevent access by a user more than once a day. As a result, most users become forever fascinated with COOKIE. At the installation where COOKIE was first implemented, the two dozen or so users on the system have been consistently "eating" COOKIEs once a day for over a year!

The COOKIE program requires two files, named COOKIES and ATE, in order to operate. The file named COOKIES contains the fortunes from which the program makes a random selection. Each item in the file is a separate fortune. The item identifier is some integer greater than or equal to 1. The rest of the item consists of only attribute one, which contains the text of the fortune itself. The easiest way to fill and maintain the COOKIES file is to use the Editor to create items for the file. The more items in the COOKIES file, the less often a user will see or remember a repeated fortune.

The ATE file is where the COOKIE program records the last date each user has read a fortune. The items in the ATE file are automatically generated and maintained by the program, and should never need editing or adjustment. The identifier of each item in ATE is the name of an account a user logged on with. The rest of the item consists only of attribute one, which contains the date, in internal form, that the account used the COOKIE program to display a fortune.

Line 1 of the program is an EQUATE that indicates how many items (fortunes) are in the COOKIES file. In this case, the EQUATE promises that COOKIES contains 500 items, which should be numbered contiguously from 1 to 500. If the COOKIES file is constantly growing because of the addition of new fortunes, it may be better to store the count of total fortunes in some file, which is then read by the COOKIE program. That way, just the count can be edited in the file each time a new fortune is collected, and the program itself doesn't have to be edited and recompiled.

Line 6 determines the name of the account invoking COOKIE. If many users share the same account, it may be desirable to change the conversion to "G 1" to cause ATE item identifiers to be port numbers. That way, COOKIE will show a fortune any number of times per day for each account, but only once per day at a given port.

Line 20 displays the fortune text, without regard to formatting of any kind, so that fortunes longer than the display width will wrap around on the screen. Some collectors of fortunes for the COOKIES file may like lengthy fortunes, in which case storing a fortune's text as multiple values or attributes in each COOKIES item, and using a LOOP at line 20 to display each text line, would be necessary enhancements. Note that in this version of the program, and assuming the user doesn't mistake the message from line 17 as a fortune, an "empty" (nonexistent) cookie allows the user another chance at a fortune, since the ATE file isn't updated before the STOP in line 18. To not allow another chance, the program could be changed by setting the COOKIE variable equal to the string in line 17, and deleting line 18.

COOKIE is most entertaining and effective at an installation with a large number of end users and with a file containing a

large number of fortunes. Besides actual fortune cookies themselves, there are endless other sources of good fortunes worth collecting. Here are a few to start your file with: GET A PERFECT CUP OF COFFEE FROM A COPPER COFFEE POT. I KNOW A LOT OF CUTE SAYINGS, BUT NONE COME TO MIND RIGHT NOW. NEVER GO TO A DOCTOR WHOSE OFFICE PLANTS HAVE DIED. QUESTION AUTHORITY. STRIVE TO LOOK TREMENDOUSLY IMPORTANT. WHEN YOU DIAL A WRONG NUMBER, YOU NEVER GET A BUSY SIGNAL. HE WHO IS IMPATIENT WAITS TWICE. ON A CLEAR DISK, YOU CAN SEEK FOREVER. THE OBSCURE YOU WILL SEE EVENTUALLY; THE COMPLETELY OBVIOUS TAKES LONGER. USE J. PAUL GETTY'S FORMULA FOR SUCCESS: RISE EARLY, WORK LATE, STRIKE OIL. THE TIME IS RIGHT TO MAKE NEW FRIENDS. REVERBERATIONS ARE OCCURRING AS A RESULT OF YOUR LIFESTYLE ADJUSTMENTS. THE ONE TIME TODAY THAT YOU LEAN BACK AND RELAX WILL BE THE ONE TIME THE BOSS WALKS BY. CAPRICORN, TAURUS AND VIRGO PERSONS ARE NEARING. DEMONSTRATE YOUR STRENGTH OF POSITION AND CONVICTIONS. APRIL BLIZZARDS WILL FREEZE YOUR GIZZARDS. TEAMWORK IS ESSENTIAL — IT WILL ALLOW YOU TO BLAME SOMEONE ELSE.

Ⓟ

Some New Subscribers

Steven Davies-Morris
Monitor Labs Inc
San Diego, CA

Thomas F. Dowling
Benefit Systems Inc
Baltimore, MD

Don Katzman
Foss Inc
Denver, CO

Peggy Morrison
RN Home Care
Moorestown, NJ

```
001 EQU TOTAL.COOKIES TO 500
002 *
003 OPEN "COOKIES" TO COOKIE.FILE ELSE STOP "ERR1"
004 OPEN "ATE" TO ATE.FILE ELSE STOP "ERR2"
005 *
006 ACCOUNT = OCONV(OCONV("","U50BB"),"G1 1")
007 TODAY = DATE()
008 *
009 PRINT
010 READU ATE FROM ATE.FILE, ACCOUNT ELSE ATE = ""
011 IF ATE = TODAY THEN
012   PRINT "YOU ALREADY ATE TODAY, ":ACCOUNT:"!"
013   STOP
014   END
015 CHOICE = RND(TOTAL.COOKIES)+1
016 READ COOKIE FROM COOKIE.FILE, CHOICE ELSE
017   PRINT "THIS COOKIE IS EMPTY!"
018   STOP
019   END
020 PRINT COOKIE
021 ATE = TODAY
022 WRITE ATE ON ATE.FILE, ACCOUNT
023 STOP
024 *
025 END
```

COOKIE PROGRAM LISTING

Ⓟ

What is COMPU-SHEET?

A new financial planning program that allows you to explore the effects of your decisions before you put them to work.

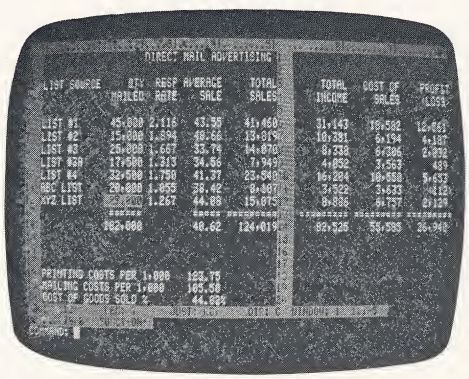
COMPU-SHEET is a powerful electronic worksheet which can be used by anyone...even without programming experience. COMPU-SHEET gives you the ability to solve, in seconds, involved problems which might take hours to do by hand. With COMPU-SHEET you can create virtually any size and type of worksheet. You can enter headings, numbers, formulas, retrieve data from other files, and then COMPU-SHEET will calculate answers based upon your worksheet design. You can scroll over the entire worksheet in any direction. COMPU-SHEET is a visible programmable calculator with the flexibility a larger computer can offer. COMPU-SHEET brings the power of the computer to your imagination and fingertips.

How can I use COMPU-SHEET?

The uses of COMPU-SHEET are limited only by your imagination. It can be used for cash flow projections...pro-forma statements...budget analysis ...job costing...estimating... advertising analysis...proposals ...investment analysis... sales forecasting...it can even be used for bar graphing. The list is endless. Then, when your worksheet is created, you can start analyzing with "WHAT IF...?" "What if sales increased...?" "What if the response rate is only...?" "What if the interest rate is ...?"

How does COMPU-SHEET work?

COMPU-SHEET is easier to use than pencil and paper. First, you lay out your worksheet by describing the width and justification of each column. Then you enter the body of the worksheet — headings, numbers, and formulas. The cursor moves from location to location or you can move the cursor up, down, left, right, or diagonally. You can make a change to any location and COMPU-SHEET will recalculate your entire worksheet in a matter of seconds. Your worksheet can be printed in whole or in part at any time. **COMPU-SHEET is powerful, flexible, and very easy to use.**



Once your worksheet is set up, you can start experimenting with "What if?" Change any number or formula and COMPU-SHEET will recalculate your worksheet.

Some of the features of COMPU-SHEET:

- Worksheet size up to 32,267 characters...allows for any number of columns and rows.
- Columns may be set to any width desired...the width of any column may be changed independently of others.
- Any number of vertical "windows" may be set...windows may scroll up, down, left, right, or diagonally.
- New columns and rows can be inserted at any point...any column or row can be deleted. Formulas are adjusted automatically.
- The worksheet may be written to a file where any other program can read or write on the worksheet.
- In-screen prompting for data ...the cursor moves in any direction over the worksheet.
- Formulas provide for standard algebraic format plus many additional operations.
- Formulas provide for reading and writing to any other system file.
- Worksheets can be merged and data can be transferred between worksheets.
- Does not use any assembly language code (user-modes).
- Operates on Microdata, CDI's Series/1, ADDS Mentor™, Ultimate™, Evolution, Prime Information™ and other PICK operating system computers.
- COMPU-SHEET is available for a one-time license fee of $795.00 with a 21-day trial period.

COMPU-SHEET brings the power of the computer to your imagination and fingertips. To learn more about COMPU-SHEET, call (602) 993-3579 or mail the form below to:

INTERACTIVE SYSTEMS
129 East Voltaire Avenue
Phoenix, Arizona 85022

I'm interested in learning more about COMPU-SHEET.

Name _____

Company _____

Address _____

City _____

State _____ Zip _____

Phone _____

Selected by CDI to be included with every IBM Series/1.

Continued from page 30

both on the same wave length.

Pragma: Is salary administration based on a formula tied into appraisals and reviews?

Kempter: Once again, we might be a little unique. We have all the employees on a salary basis, but I also attach a bonus to the various customized software applications that we do for our clients, and if the work is chargeable, I will give the programmers a certain percentage of the amount we're charging our clients if the project is completed on time. So it's a salary plus a target incentive type bonus program that we operate off of. And I think that it's really good, because as soon as they do something like complete a project on time, they're immediately rewarded versus waiting until a year is over and they forget about it. Plus it really motivates them to concentrate on that one project and get completed on a timely basis.

Pragma: How do you determine the data processing budget?

Kempter: We do it on an annual basis. It's really pretty simplistic to do since we only have basically two types of costs, and that's our hardware and our people. Those two probably give you 80% or 90% of the cost that will incur.

Pragma: Are your people involved with local user groups?

Kempter: Yes, we belong to a local user group in the Delaware Valley, and we belong to Micru International. We just joined IDBMA about a month ago.

Pragma: For all this user group activity, what rewards to you get?

Kempter: Oh, cross fertilization of ideas. See what other people are doing with their Microdata systems. It's good exposure for our data processing staff to throw around common problems that we're incurring with our systems and hardware. I find it very rewarding. And there is strength in numbers in dealing with Microdata.

Pragma: Are you going to pay more attention to other compatible systems in the future?

Kempter: Definitely. We have as one of our goals to convert within the next three year time period all our packages that are written in PROC to DATA/BASIC, and that's the responsibility of one of our programmers. I definitely want to look hard at other vendors in the future. I hope we grow, I hope we expand. But I don't want to be strictly tied down to Microdata. I want to look at Prime, or Ultimate, or ADDS Mentor, or whatever else is available. I think it gives you better bargaining power with your vendors when you're out there buying hardware and software. Plus, I have a dream that possibly we could take a lot of the software we have developed and make it compatible with various other pieces of hardware. We possibly could sell this to other people.

Pragma: Does your company make use of microcomputer systems at all?

Kempter: No. We're looking at them, though. One thing we're trying to do is integrate an Apple microcomputer with our Microdata system. The reason we want to do that is to be able to use Visicalc and other software packages like that. We worked with the people from Apple, but we hit a few problems and snags in being able to transfer the data back and forth, so now that project is on hold status. We are looking at the Radio Shack, we're looking at what other CPA firms are doing. I understand the Big 8 firms are using Osborne microcomputers, to actually take out with them on jobs. Microdata has told us that their new Prism 4 terminals are really microcomputers and that all the guts are there, they just need the software to make them work. So we think there's a definite place for them, we're just not sure exactly how it's going to fit into what we're doing. But I see it possibly as a means of offloading a lot of the data off of our mini, capturing it at the source and at some point transferring it downstream to our computer.

Pragma: Galinski Hamburg has come a long way in data processing. What do you think will happen in the next few years?

Kempter: I'd like to see us be able to, in the next two to three years, double what we have right now, as far as the number of clients on the system and the dollar amount of revenue that we're generating. I hope that, in the next two or three years, we'll be able to go out and purchase another Sequel or whatever type computer. I think it's unique what we're selling. I think no one else is doing it at the present time, and I think the market is gigantic. I think they're just waiting for us to knock on their doors.

P

Continued from page 12

The design of PROFILE is very similar to TRIM.DELIM, in that the program is basically three nested FOR loops. The code begins by OPENing the file of interest and then asking if the operator wants to refresh the display after each item. If refreshing is chosen, the operator can observe PROFILE's report being continuously updated as the file is being scanned, but it will take much longer to finish scanning the file. If refresh is not used, the program scans through the file faster, but the report is not output until after the last item has been read.

Line 7 initializes what will be data for five of the report columns. MAX.AMC in line 8 will be the largest AMC encountered. Line 16 effectively counts the items found (even if the item identifier is null), and lines 17 and 18 update MIN and MAX ID for the first report row.

Lines 20 to 44 were inspired by the TRIM.DELIM program, with some slight modifications (the adding of the .TXT variables) in an attempt to improve item scanning time. Note that line 25 bumps the FOUND count for an AMC even if only a system delimiter (not actual data) has been found in the attribute, and that line 32 excludes null values from MIN tests. Lines 33 to 40 update the MIN and MAX columns once a value has been extracted from the item. The output subroutine beginning at line 48 is called either from line 13 if the report is only to be output once at the end of all processing, or it is called from line 45 if output is to be refreshed after scanning each item. Note that if the number of attributes (output rows) exceeds the number of available lines on the display screen, the output will scroll, making the refresh option unpleasant and requiring output to be sent to a printer unless some form of display paging is used.

PROFILE inflexibly uses fixed field widths and justification that may be unsuitable for a given file. Perhaps the program should read a file's dictionary to help decide the best format for outputting each of the final report rows. Another possible enhancement would be to modify PROFILE to include a seventh report column showing a single code character such as A, V or S to indicate whether the data stored in each attribute includes only attributes, or some values, or some subvalues.

Even without any enhancements, both TRIM.DELIM and PROFILE can be valuable utilities, worth including in an installation's software toolbox. Ⓟ

Some New Subscribers

John H. Hardman
Hardman Inc.
Belleville, NJ

Ward T. Vuillemot
Detection Systems Inc.
Fairport, NY

Serge Krisiuk
Brentwood Data Ltd.
Mississauga, ON

Leo T. Fohl
Stanley Air Tools
Cleveland, OH

Continued from page 27

data for each component part? *In Vanilla, one alternate part number may be included as extra data for each component part.*

LINE ITEM FORMAT. Is each line item in the list of bill components in the same format, and equivalent to the data for one part in the assembly? Or are arbitrary notes and other miscellaneous line items interspersed with part data? *In Vanilla, if a component part number is not entered for a line item, the line item's description field (normally implied by the component part number) may be used for notes or arbitrary data, thereby allowing special line items.*

OTHER COMPONENT ATTRIBUTES. These are typically things like reference designators (for cross-referencing the location of components on schematics), item numbers (for sequentially numbering components to ease searching when parts are not in sorted order), and other bits of information of interest to Engineering personnel. *Although easily modified to include such data, for now Vanilla maintains only the part number, description, quantity and alternate for each component. As a bill is being updated, Vanilla does display the relative number of the current line item being shown.*

OTHER BILL ATTRIBUTES. These are typically things like revision and status information, where-used indicators, page counts and sizes for the actual equivalent paper documents, and other bits of information of interest to Engineering personnel. *In Vanilla, a notes field consisting of any number of 50 character lines is provided for capturing such data.*

The mention of where-used indicators brings up the subject of Vanilla's bill of material file structure. In most software systems for maintaining bills of material, the bill of material input program will not only keep the bill of material file updated, but it will also maintain an inverted version of the bill of material file, called the where-used file. The where-used file allows tracing a product's structure from the bottom up, so that it can quickly and easily be determined in what assemblies a given part is used. Strictly speaking, a where-used file is only necessary if an efficient "multi-level" where-used capability is required. (A multi-level where-used report shows all higher assemblies that use a given part, regardless of at what level the part is actually used.) If the only where-used requirement is to report at what immediately higher (single) level a part is used, then an inquiry language like ENGLISH can be used to simply search the bill of material file for any occurrence of a given part, and a where-used file is not really necessary. Such a brute force search will be much slower than a simple lookup in a where-used file, but since the frequency of use of a where-used capability if often quite low, the cost of developing and maintaining where-used software and data can easily overrun the cost of occasional delays while waiting for a search of the bill of material file to finish. Because ENGLISH can easily search the bill of material file for single level where-used occurrences, and since multi-level where-used reports are more of a luxury for Engineering personnel than a necessity for a working MRP system, Vanilla does not bother to maintain a where-used file.

The internal structure of a bill of material file can be as varied as the number of vendors of manufacturing software. The bill of material system in one manufacturing package out of the Seattle area was inspired by an earlier IBM collection of software, so it chose to structure its bills of material so that each item in the file corresponds to one component part in a bill, with pointers to the preceding and following components to form linked lists, both across and up and down the product

GET.BM SCREEN DEFINITION

```
001
002
003 ID]REV]DESC]SITEM]PART]SREV]CDESC]QTY]SUM]ALT]
    NOTE]OK]SPART]SDESC]SQTY]SALT]SNOTE
004
005 \\\\          Bill:          Revision:\\   Des
    cription:\\\          Item:\\   1 Component:
          Revision:\\ 2 Description:\\    3 Quan
    tity:                U/M:      4 Alternate par
    t:\\       5 Note:
006 0,0
007 ]]]]]]]]]]]]PART]CDESC]QTY]ALT]NOTE
008
009 S]S]S]S]S]S]S]S]S]S]S]S]N]N]N]N]N
010
011 L]L]L]L]L]L]L]L]L]L]L]L]L
012 1]1]1]1]1]1]1]1]1]1]1]1]1
013 Y
014
015
016 4,16]4,35]6,16]9,16]11,16]11,35]13,16]15,16]15
    ,36]15,59]17,16]22,37
017 7]1]30]5]7]1]30]10]2]7]50]2
018
019 ]TPM;X;;1]TPM;X;;2
020 7]1]30]5]7]1]30]10]2]7]50]2
021 ]1]1]1]1]1]1]1
022 Bill of material number (up to 7 digits):]]]]C
    omponent part number (up to 7 digits):]]Item d
    escription (any 30 characters):]Quantity (nume
    ric):]]Alternate part number (up to 7 digits):
    ]Note:]File (FI) or exit (EX) or goto (1-5):
023 22,0]]]]22,0]]22,0]22,0]]22,0]22,0]22,0
024 22,41]]]]22,39]]22,37]22,19]]22,39]22,5]22,37
025 39]]]]41]]43]6]]]41]75]43
026
027 7]]]]7]]30]]]]50]2
028
029
030
031 MD0]]]]MD0]]]]]MD0
032 ]]F]F]F]F]F]F]F]F]F]F]F]F]F]F]F
033
034
035
036
037
038
039
040 ('EX')E
```

structure. The advantage of this approach is that any limitations potentially imposed by some operating system on the maximum record size allowed is never a problem, since only one component (not the whole bill of material) is ever stored in a file record. The disadvantage of this approach is that listing a complete bill of material requires a program written to deal with all those pointers, since interpreting the structure of such a file is typically beyond the capabilities of a report generation language like ENGLISH. Another package out of Southern California chooses to maintain its bills of material as just one record, which simplifies many of the programs that have to deal with the file. But that software also chooses to store each component as a separate attribute (not value!) in the file item, which still makes reporting with ENGLISH hopeless. In Vanilla, all the data for a bill of material is stored in one file item, and the data for each component part in the bill is saved as an associated group of values, to aid in report generation with ENGLISH.

In any software system consisting of a collection of programs, it is desirable to maintain a certain consistency in the appearance and operation of the programs, so that the problems of operator documentation, training and understanding are simplified. Vanilla's bill of material input program (see the GET.BM program listing beginning on page 28) is designed to provide and enforce a number of conventions regarding operator inputs, and these conventions will continue to reappear in other, future Vanilla software. Some of these conventions are:

FORMATTED SCREENS. Vanilla programs use formatted screens with fixed field locations for each piece of data that is input or displayed. Whenever an existing item is called up for editing, all current data is first displayed to give the operator a complete picture of the state of the item. If an error message is output at any time, it is always displayed at the lower left hand corner of the screen. And since many file records are often designed to contain an arbitrary number of line items consisting of an associated group of attribute values, but there is limited space on the display screen, only one line item at a time is every displayed, and mechanisms (such as hitting the RETURN key) are provided to allow the operator to cycle through all line items and display any particular item of interest. (Vanilla does not rely on SCREENPRO's multivalue display and update method using the LINE FEED key, because it is too limited for many types of applications.)

EXIT COMMAND. At any time, the operator may input "EX" to either exit from the group of fields being input to get to the final file-or-exit prompt, or exit the current file item (from the file-or-exit prompt), or exit the entire program (from the initial prompt for an item identifier).

GOTO COMMAND. From the final file-or-exit prompt, the operator may go back to a field and resume responding to all prompts for data.

FIELD CLEARING. Any non-required field can be cleared of data with the backslash key. (This is a SCREENPRO convention.)

SEPARATE DELETES. Programs separate from the main input program for any file are used for deleting file items, to prevent unskilled operators from accidentally destroying data.

In the same way that Vanilla programs all support a number of conventions for the operator, the DATA/BASIC code also follows certain conventions for the programmer. For example, all attribute references are symbolic, with the variable names all appearing as EQUATEs at the start of each program in the form FILE$ATTRIBUTE. And all SCREENPRO step references appear as lower case EQUATEs. More of these conventions will be discussed as more Vanilla code is presented.

The code in GET.BM may seem to have a preponderance of statement labels, but there is really a very small number of GOTOs (a concession from structured programming techniques in the interest of allowing easy data entry procedures for the operator), and so the program is actually straightforward and easy to follow. In fact, the code can easily serve as a model for any application with a major requirement of supporting the input of an associated group of attribute values: modifying GET.BM to become some other program for maintaining a file that happens to have items structured similar to bills of material should be an easy task.

Note that since a bill's revision and description are displayed only once by line 34, translate conversions in the SCREENPRO screen definition are used to do the job. But since any number of component revisions, descriptions, and units of measure might be displayed during the update session for one bill of material, the program bothers to load that data itself (lines 43 to 55). Also, a component's part number and description must both be deleted in order to delete a line item (line 83). The program's weakest point is its inability to allow new line items to be inserted ahead of other line items. Such a feature can be added by including an INsert command option at the file-or-exit prompt. The option would transfer to line 59, while making sure to INSert a group of nulls with each input of a new part, to make room in the bill of material record. Any code that exits the line item creation loop would then have to make sure at least a part number or description was present in a line item, otherwise the hole left by the INSertion attempt would have to be DELeted.

In the next installment, Vanilla will venture into another major source of file data for MRP: purchasing.

P

Some New Subscribers

Arthur F. Huber
Nolts Pond Inc.
Silver Spring, PA

Mark Nelson
DMS of Green Bay
Green Bay, WI

Malia Culbreth
Colorado Vision Services Inc.
Arvada, CO

Joseph T. Ameres
Charles P. Young Co.
New York, NY

CARGO

A Spread Sheet Generator for Microdata Computers!

- MARKET SHARE ANALYSIS
- CAPITAL BUDGETING
- DISCOUNT SCHEDULES
- CASHFLOW
- COMMISSION PLANS
- PRODUCT PRICING
- LONG RANGE PLANNING
- DIVISION SUMMARIES
- CAPACITY PLANNING
- INVENTORY FORECASTS
- PRO FORMAS
- JOB COST ESTIMATING
- INCOME STATEMENTS
- SALES FORECASTS
- "WHAT IF" MODELING
- BALANCE SHEETS
- MAKE OR BUY OR LEASE
- PROFIT & LOSS STATEMENTS
- REQUIREMENTS PLANNING
- CONSOLIDATIONS
- ANY SPREAD SHEET APPLICATION

What is Cargo?

Cargo is powerful software — a program — for Microdata computers that allows users with no computer training to quickly and easily create, compute, manipulate, display and print spread sheets...spread sheets such as profit and loss statements, sales forecasts, budgets — any of the kinds of tabular reports found throughout a business organization. If you are working with columns and rows of numbers and text, Cargo will automate your work — without the help of programmers or other data processing personnel!

Why Use Cargo?

Cargo makes spread sheet revisions and turnaround extremely quick and easy • Cargo avoids programming, especially those expensive program development and maintenance costs • Cargo allows non-programmers to implement useful business applications on the computer • Cargo quickly pays for itself • If desired, Cargo can be incorporated into ongoing development work by programmers, allowing current programming projects to be completed sooner • Cargo output is self-documenting • Cargo completely replaces the necessary programming in most spread sheet applications.

How Does Cargo Work?

To create a spread sheet with Cargo, a user begins by typing into the computer the list of column and row names that border the sheet. Typically, the columns are time periods such as JAN, FEB, MAR...while the rows are categories such as INCOME, SALES, INVENTORY, and so on. Next, the user inputs the starting values for various positions on the sheet. Then, using a simple, menu-driven, "fill-in-the-blanks" approach, the user defines the rules by which the rest of the sheet values should be computed. For example, *Add all rows from row FOOD through row SHELTER put result in row EXPENSES.* At this point, all necessary data for the sheet has been supplied. Cargo always has a current version of the sheet on file, so the user never has to repeat any of the work done to develop a spread sheet. The user can now command Cargo to calculate the remaining values on the sheet and output the final results. If desired, the user may specify parameters to cause the final report to appear in any desired format. Often, the user will want to modify some of the various columns, rows, values or calculation rules and then recompute the complete sheet. This can quickly and easily be done any number of times with Cargo, allowing the user to experiment with report formats, test assumptions about initial or final data values, ask "what if..." questions for planning — in short, use a powerful tool to communicate results, examine alternatives, and improve decision-making.

Features

Designed and offered exclusively for Microdata systems • Totally on-line and interactive • Fully menu driven with continuous prompting • Simple to understand • Easy to use • No prior computer knowledge required • Field proven • No special syntax or formulas to memorize • Complete documentation, all on-line and immediately accessible • Includes step-by-step demonstration examples • Installs in seconds • One copy of Cargo simultaneously supports any number of terminals, accounts and sheets • Absolutely no assembly language code, special user modes, or hardware modifications for insured compatibility with future operating system releases • Fully symbolic: names instead of numbers reference sheet columns and rows (*Add WAGES with DIVIDENDS put result in INCOME* instead of cryptic methods like *ROW 3 = ROW 1 + ROW 2)* • Full complement of sheet calculating operations • Number of columns and rows limited only by Microdata file and record capacities (total column and row name lists may not exceed 32,267 characters) • Cargo has been used for spread sheets with hundreds of rows and columns in each sheet • All data structures transparently maintained as files — simplifies use of Cargo by programmers, yet allows users to totally ignore file system • Defaults used for all parameters to simplify initial sheet creation • Repetitive data easily created by input commands or by calculation • File data can be read into sheets • Multiple sheets easily consolidated • Automatic rotation of time period columns for rolling reports • Sheet copy capability • Portions of sheets can be selected for printing • Calculation results displayable before printing • Sheets can be printed as data capture worksheets, data entry validation sheets, or final report sheets • All calculation rules printable as easy-to-read prose • Printed report widths, spacing, underlining, decimal places, headings, footings and other parameters all under user control.

How To Try Cargo

1. Fill out, sign, and date the License Agreement below. Be sure to include your mailing address, SYSTEM SERIAL NUMBER, and signature. Do NOT use a Post Office box, if possible.
2. Circle your tape drive density: (800 BPI) or (1600 BPI)
3. Enclose payment of $880. California buyers must add 6% sales tax.
4. Send this order, the License Agreement, and your payment to: Semaphore Corporation
   207 Granada Drive
   Aptos, CA 95003
5. Loading instructions and a complete Cargo magnetic tape, which includes all documentation, will be delivered to you within 15 days of receiving your order.
6. If within 30 days of receiving your tape you are not completely satisfied with Cargo for any reason, we will permit you to return Cargo, and we will then return your uncashed check to you.

Cargo is available immediately, but currently only for computers using Microdata operating system release 4.1 or later • Since Cargo is menu driven and uses formatted screens extensively, 9600 baud terminals are recommended (but not required) when using Cargo • Quantity discounts available • Dealer inquiries invited • Telephone Semaphore Corporation at (408) 688-9200.

Non-Exclusive Software License Agreement

Semaphore Corporation will provide a non-exclusive perpetual license to
Company Name _____
Address _____
_____
_____
(hereafter referred to as "Licensee") to use the Cargo software (hereafter referred to as "Software"). The Licensee hereby agrees the Software being licensed, and all alterations, modifications, revisions and enhancements thereto, whether now or hereafter made, constitute proprietary information, rights and trade secrets of Semaphore Corporation, and that title and full ownership rights to the Software and to any alterations, modifications, revisions and enhancements thereto, shall remain in Semaphore Corporation. The Licensee hereby agrees the Software shall be used only by the Licensee on the Licensee's Microdata computer equipment, mainframe serial number [_____]. The Licensee hereby agrees not to resell, trade or otherwise make the Software available in any form to any other person, persons or company without prior written permission from Semaphore Corporation. The term of this license shall commence upon the signing of this agreement and shall remain in force perpetually unless terminated by Semaphore Corporation. Semaphore Corporation makes no warranty, express or implied, concerning the applicability of this Software to any specific purpose. It is solely the Licensee's responsibility to determine suitability of the Software for a particular purpose. Semaphore Corporation accepts no liability for loss or damage caused, or alleged to be caused, directly or indirectly, by any Software sold by Semaphore Corporation, including but not limited to any interruption of service, loss of business or anticipatory profits or consequential damages resulting from the use or operation of such Software. This statement of limited liability is in lieu of all other warranties or guarantees, express or implied, including warranties of merchantability and fitness for a particular purpose.

Signed by _____ Date _____ Telephone _____
Printed Name _____ Title _____

PRAGMA
207 GRANADA DRIVE
APTOS, CA 95003

BULK RATE
U.S. POSTAGE
PAID
APTOS, CA 95003
Permit No. 67

ADDRESS CORRECTION REQUESTED